Kapitel 1
Einführung

In diesem Kapitel lernen Sie erste VBA-Makros kennen. Sie erfahren, wie sie aufgebaut sind und wie Sie sie verändern. Außerdem werde ich die VBA-Entwicklungsumgebung erläutern.

Die Abkürzung VBA steht für *Visual Basic for Applications*. Es handelt sich dabei um die Programmiersprache *Visual Basic* mit passenden Ergänzungen für die verschiedenen Anwendungen in Microsoft Office.

Mit Microsoft Excel werden viele alltägliche Aufgaben im beruflichen und privaten Bereich bereits hervorragend bewältigt. Durch eine zusätzliche Programmierung mit VBA geht das bedeutend schneller. Zudem werden viele Aufgaben erst dank VBA gelöst.

Sie erlernen anhand von leicht verständlichen, schrittweise aufgebauten Beispielen, wie Sie VBA als Ergänzung zu Excel sinnvoll einsetzen. Eigene Übungen (mit Lösungen im Anhang) unterstützen Sie bei der Überprüfung Ihres Wissensstands.

Für die Hilfe bei der Erstellung dieses Buches bedanke ich mich beim ganzen Team des Rheinwerk Verlags, ganz besonders bei Anne Scheibe.

1.1 Was wird besser durch Makros und VBA?

Ein Makro besteht aus einer Reihe von Anweisungen, die nacheinander ausgeführt werden. Durch jede dieser Anweisungen wird in Excel ein bestimmter Vorgang ausgeführt, z. B. eine Zahl in eine Tabellenzelle geschrieben oder die Farbe eines Diagrammbalkens geändert.

Die Anweisungen sind in der Sprache *VBA* geschrieben. VBA-Programme gehen über solch einfache Makros weit hinaus und steuern komplexe Abläufe. In der alltäglichen Arbeit wird Excel von zwei Gruppen genutzt:

- von Excel-Benutzern, die sich mit einfachen Themen wie z. B. der Dateneingabe und dem Aufruf von Excel-Anwendungen befassen

- von erfahrenen Excel-Programmiererinnen, die sich mit der Entwicklung von Excel-Anwendungen zur Unterstützung der Excel-Benutzer befassen

Die Entwicklung der Excel-Anwendungen kann im selben Unternehmen stattfinden. Es kann sich aber auch um eingekaufte Excel-Anwendungen handeln. Es folgen einige typische Szenarien, die die Nutzung und die Vorteile von VBA zeigen:

- Es werden große Mengen an Daten aus einer Textdatei eingelesen. Nur der Aufbau der Daten ist bekannt, nicht aber die Menge. Außerdem werden, abhängig vom aktuellen Inhalt der Textdatei, nur bestimmte Daten gelesen. Nach dem Einlesen werden die Daten verarbeitet, formatiert, zusammengefasst und zur Verdeutlichung grafisch dargestellt.
- Es wird ein Diagramm aus einer Tabelle erstellt. Die jeweilige Größe und der Inhalt der Tabelle bestimmen die Art des Diagramms und seine Darstellung.
- Benutzer haben nach dem Aufrufen der Excel-Anwendung ein Dialogfeld vor sich. Darin nehmen sie, abhängig von der jeweiligen Situation, bestimmte Einstellungen vor, treffen die gewünschte Auswahl der Daten und starten dann deren weitere Verarbeitung.
- In regelmäßigen Zeitabständen ergeben sich Daten, die auf eine ähnliche, aber nicht identische, Art und Weise weiterverarbeitet werden.
- Die Verarbeitung bestimmter Daten ist komplex und erfolgt in mehreren Schritten. Mit VBA vereinfachen Sie die Koordination und die Durchführung der einzelnen Schritte.
- Benutzerinnen steht nach dem Aufruf von Excel nur eine bestimmte Funktionalität zur Verfügung. Excel kann durch VBA gleichzeitig:
  - eingeschränkt werden, wodurch weniger Fehler gemacht werden;
  - erweitert werden, wodurch besondere Funktionen und Abläufe zur Verfügung stehen, die über Excel hinausgehen.
- Zur Bewältigung der laufenden Geschäftsprozesse werden bereits Excel-Anwendungen eingesetzt. Nach einer Änderung der Prozesse werden die Excel-Anwendungen entsprechend angepasst und gegebenenfalls erweitert.

In diesem Buch werde ich verschiedene Möglichkeiten zur Problemlösung mit VBA anhand von Beispielen aus der Praxis zeigen. Dabei werden wir mit einer umfangreichen Auswahl aus der Sprache VBA und der zugrundeliegenden Excel-Objektbibliothek arbeiten. Das ermöglicht Ihnen, selbständig mit Excel und VBA zu arbeiten, ohne den Überblick zu verlieren.

1.2 Mit Makros arbeiten

Zu Beginn erstellen wir ein einfaches Makro, das anschließend ausgeführt wird. Hinterher werfen wir einen ersten Blick auf den VBA-Code.

Das Thema *Makrosicherheit*, also die Sicherheit vor fremden Makros, die schädlichen Code enthalten können, spielt eine große Rolle. Diese Thematik werde ich in Abschnitt 1.2.5, »Makrosicherheit ändern«, erläutern.

Excel-Dateien können sowohl mit als auch ohne Makros gespeichert werden. Darauf werde ich in Abschnitt 1.2.4, »Makro speichern«, eingehen.

Die in diesem Kapitel erläuterte Bedienung gilt für Excel 2016 bis Excel 2021. Die weitaus meisten Beispiele in diesem Buch laufen auch unter noch älteren Versionen. Sämtliche Beispiele finden Sie in den Materialien, die Sie über *www.rheinwerk-verlag.de/5546* erreichen.

1.2.1 Makro aufzeichnen

Mithilfe der folgenden Beschreibung erstellen Sie ein einfaches Makro zur Verschiebung des Inhalts von Zelle `A1` in Zelle `C1`. Das Beispiel dient einer ersten Verdeutlichung der Abläufe, noch ohne VBA:

1. Starten Sie Excel.
2. Sie haben eine leere Arbeitsmappe vor sich. Tragen Sie in Zelle `A1` einen beliebigen Inhalt (Zahl oder Text) ein (siehe Abbildung 1.1).

Abbildung 1.1 Zelle, deren Inhalt verschoben wird

3. Betätigen Sie auf der Registerkarte ANSICHT den unteren Teil der Schaltfläche MAKROS, der mit einem Pfeil gekennzeichnet ist. Es klappt ein kleines Untermenü auf, siehe Abbildung 1.2.

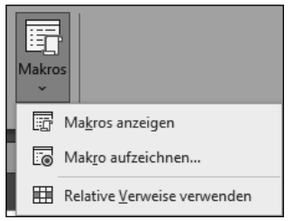


Abbildung 1.2 Menü »Ansicht • Makros«, Untermenü

4. Wählen Sie den Menüpunkt MAKRO AUFZEICHNEN. Es erscheint das Dialogfeld MAKRO AUFZEICHNEN (siehe Abbildung 1.3). Wählen Sie in der Liste MAKRO SPEICHERN IN: den Eintrag DIESE ARBEITSMAPPE, falls diese Option noch nicht eingestellt ist.

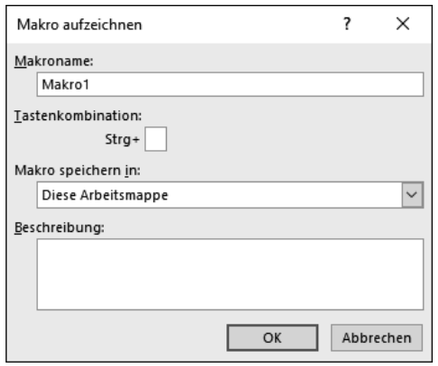


Abbildung 1.3 Makro mit dem Namen »Makro1«

5. Der vorgeschlagene Makroname lautet `Makro1`. Diesen können Sie beibehalten.
6. Betätigen Sie die Schaltfläche OK. Ab jetzt werden alle Aktionen, die Sie ausführen, aufgezeichnet.
7. Wählen Sie Zelle `A1` aus.
8. Schneiden Sie den Inhalt von Zelle `A1` aus, zum Beispiel mit Strg+X.
9. Wählen Sie Zelle `C1` aus.
10. Fügen Sie den Inhalt von Zelle `A1` ein, zum Beispiel mit Strg+V.

11. Damit wurde der Inhalt von Zelle `A1` in Zelle `C1` verschoben (siehe Abbildung 1.4).
12. Klappen Sie in der Registerkarte ANSICHT wiederum über den Pfeil auf der Schaltfläche MAKROS das Menü auf.
13. Wählen Sie den Menüpunkt AUFZEICHNUNG BEENDEN. Damit wird die Aufzeichnung Ihrer Aktionen beendet.
14. Sie haben damit soeben Ihr erstes Makro erfolgreich erstellt.

Speichern Sie die Datei an dieser Stelle noch nicht.

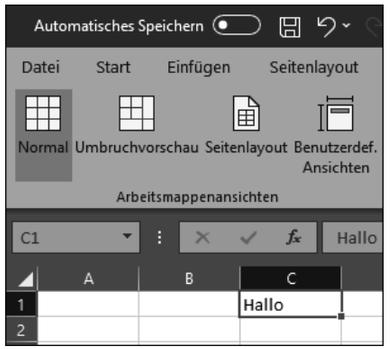


Abbildung 1.4 Zellen nach der Verschiebung

Das Aufzeichnen geht etwas schneller, wenn Sie auf der Excel-Oberfläche das entsprechende Symbol in der Statusleiste betätigen (siehe Abbildung 1.5).

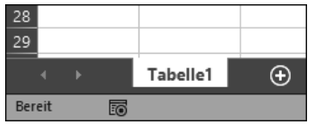


Abbildung 1.5 Symbol zum Starten einer Makroaufzeichnung

Nach Beginn der Aufzeichnung erscheint an der gleichen Stelle das Symbol zum Beenden der Aufzeichnung (siehe Abbildung 1.6).

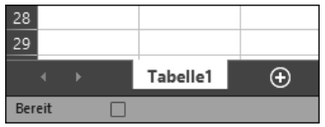


Abbildung 1.6 Symbol zum Beenden einer Makroaufzeichnung

Übung »Makro aufzeichnen«

Tragen Sie eine Zahl oder einen Text in Zelle `E3` ein. Zeichnen Sie ein Makro auf, das den Inhalt von Zelle `E3` in Zelle `E5` kopiert. Nennen Sie das Makro `KopieE3E5`.

1.2.2 Makro ausführen

Das soeben erstellte Makro `Makro1` wird nun ausgeführt. Gehen Sie dazu wie folgt vor:

1. Tragen Sie in Zelle `A1` einen beliebigen Inhalt ein (Zahl oder Text). Wählen Sie anschließend Zelle `A1` aus.
2. Klappen Sie auf der Registerkarte ANSICHT über den Pfeil unten auf der Schaltfläche MAKROS das Menü auf.
3. Wählen Sie den Menüpunkt MAKROS ANZEIGEN.
4. Es erscheint das Dialogfeld MAKRO mit einer Liste der bisher erstellten Makros. Wählen Sie in der Liste das Makro `Makro1` aus.
5. Betätigen Sie die Schaltfläche AUSFÜHREN.
6. Der Inhalt von Zelle `A1` wird wiederum in Zelle `C1` verschoben.

Damit haben Sie Ihr erstes Makro erfolgreich ausgeführt.

Übung »Makro ausführen«

Tragen Sie einen neuen Inhalt in Zelle `E3` ein. Wählen Sie anschließend Zelle `E3` aus. Führen Sie das Makro `KopieE3E5` aus, und überprüfen Sie anhand des Ergebnisses die korrekte Umsetzung.

Hinweis

Geben Sie einem Makro einen möglichst sprechenden Namen, der etwas über seine Arbeitsweise aussagt. Sie finden es dann leichter in der Liste der Makros.

1.2.3 Makro ansehen

Sehen wir uns den VBA-Code des soeben erstellten Makros an, um einen ersten Eindruck von der Sprache zu erhalten. Dazu gehen Sie wie folgt vor:

1. Klappen Sie in der Registerkarte ANSICHT über den Pfeil auf der Schaltfläche MAKROS das Menü auf.

2. Wählen Sie den Menüpunkt MAKROS ANZEIGEN.
3. Es erscheint das Dialogfeld MAKRO mit einer Liste der bisher erstellten Makros. Wählen Sie in der Liste das Makro `Makro1` aus.
4. Betätigen Sie die Schaltfläche BEARBEITEN.

Es erscheint der *Visual Basic Editor* (kurz: VBE), die eigentliche Entwicklungsumgebung. Auf seine einzelnen Elemente werde ich in Abschnitt 1.3, »Visual Basic Editor«, eingehen. Zunächst ist im rechten Fenster der Code der ausgewählten Makros zu sehen (siehe Abbildung 1.7).

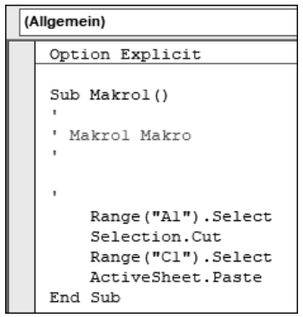


Abbildung 1.7 VBA-Code des Makros »Makro1«

Es folgt eine kurze Erläuterung der einzelnen Zeilen. Es macht nichts, falls Sie noch nicht alles genau verstehen. Die einzelnen Codeelemente werde ich später ausführlicher erklären.

Der Code des Makros `Makro1` ist in einer Sub-Prozedur zwischen `Sub` und `End Sub` notiert. Nach dem Namen eines Makros bzw. einer Sub-Prozedur werden Klammern gesetzt, die zunächst leer sind.

Das Hochkomma dient dazu, eine ganze Zeile bzw. den Rest einer Zeile zu einem Kommentar zu machen. Der Text hinter einem Hochkomma dient der Erläuterung des VBA-Codes. Er wird nicht ausgeführt.

Die Anweisung `Range("A1").Select` bedeutet: Es wird Zelle A1 ausgewählt. Dank der Anweisung `Selection.Cut` wird der Inhalt der aktuellen Auswahl, also der Inhalt von Zelle `A1`, ausgeschnitten und befindet sich anschließend in der Zwischenablage. Die Anweisung `Range("C1").Select` bedeutet: Zelle `C1` wird ausgewählt. Dank der Anweisung `ActiveSheet.Paste` wird der Inhalt der Zwischenablage in das aktuell aktive Tabellenblatt eingefügt.

Durch diesen Ablauf wird der Inhalt von Zelle `A1` in Zelle `C1` verschoben.

Schließen Sie den VBE über das Menü Datei • Schliessen und zurück zu Microsoft Excel. Es erscheint wieder die Excel-Oberfläche.

> **Übung »Makro ansehen«**
>
> Interpretieren Sie den VBA-Code des Makros `KopieE3E5`:

```
Sub KopieE3E5()
    Range("E3").Select
    Selection.Copy
    Range("E5").Select
    ActiveSheet.Paste
End Sub
```

1.2.4 Makro speichern

Excel-Dateien können sowohl mit Makros als auch ohne Makros gespeichert werden. Zum Speichern rufen Sie das Menü Datei auf und darin den Menüpunkt Speichern. Auf der rechten Seite wählen Sie die Schaltfläche Durchsuchen. Es erscheint das Dialogfeld Speichern unter.

Als Dateityp ist im Dialogfeld Speichern unter angegeben: Excel-Arbeitsmappe mit der Dateiendung *.xlsx* (siehe Abbildung 1.8).

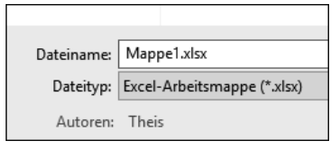


Abbildung 1.8 Als Arbeitsmappe ohne Makros speichern

Betätigen Sie nun die Schaltfläche Speichern, erscheint der Hinweis, dass die Makros in dieser Datei bei diesem Dateityp nicht mitgespeichert würden (siehe Abbildung 1.9) und daher verloren gehen würden.

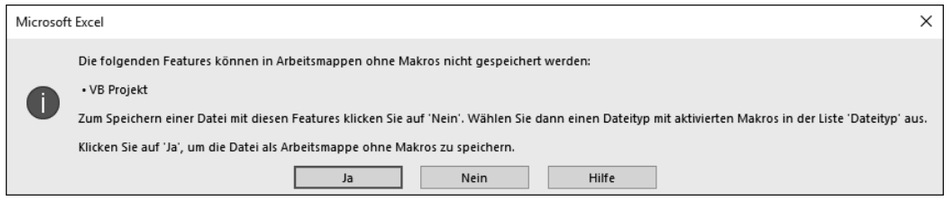


Abbildung 1.9 Warnung, dass Makros nicht mitgespeichert werden

Daher betätigen Sie zunächst die Schaltfläche Nein. Wählen Sie im Dialogfeld Speichern unter als Dateityp die Option Excel-Arbeitsmappe mit Makros (mit der Dateiendung *.xlsm*) aus, und speichern Sie die Datei unter dem gewählten Namen im gewünschten Verzeichnis, z. B. *C:\Temp\Mappe1.xlsm* (siehe Abbildung 1.10).

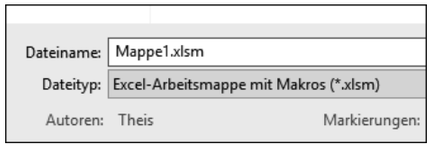


Abbildung 1.10 Als Arbeitsmappe mit Makros speichern

Nachdem die Datei erfolgreich mit Makros gespeichert wurde, schließen Sie Excel.

1.2.5 Makrosicherheit ändern

Starten Sie Excel erneut. Beim Öffnen der soeben gespeicherten Datei stoßen Sie unweigerlich auf das Thema *Makrosicherheit*. Was bedeutet das?

Excel möchte Sie davor bewahren, möglicherweise schädlichen VBA-Code auszuführen. Unterhalb der Multifunktionsleiste erscheint daher ein Sicherheitshinweis, dass die in dieser Datei enthaltenen Makros deaktiviert wurden (siehe Abbildung 1.11).

Abbildung 1.11 Sicherheitswarnung »Makros wurden deaktiviert.«

Diese Reaktion erfolgt aufgrund der folgenden Standardeinstellung für die Makrosicherheit in Excel: Alle Makros mit Benachrichtigung deaktivieren. Versuchen Sie, ein Makro auszuführen, erscheint die Information, dass dies nicht möglich ist (siehe Abbildung 1.12).

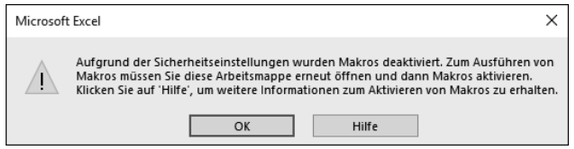


Abbildung 1.12 Hinweis, dass die Makroausführung nicht möglich ist

Sie haben die Möglichkeit, die in dieser Datei enthaltenen Makros zu aktivieren. Schließen Sie dazu die Datei, und öffnen Sie sie erneut. Betätigen Sie die Schaltfläche INHALT AKTIVIEREN neben der Sicherheitswarnung (siehe Abbildung 1.11). Die Sicherheitswarnung verschwindet, und Sie können die Makros in dieser Datei ausführen.

Die Datei wurde in die Liste der »vertrauenswürdigen Dokumente« aufgenommen. Sie können auch alle Excel-Dateien in einem Verzeichnis für vertrauenswürdig erklären, siehe Abschnitt 1.2.7, »Makrosicherheit dauerhaft ändern«.

1.2.6 Registerkarte »Entwicklertools«

Die Registerkarte ENTWICKLERTOOLS bietet weitergehende Möglichkeiten zur Erstellung und Verwaltung von Makros. Daher wird sie am besten dauerhaft in Excel eingeblendet.

Rufen Sie dazu das Menü DATEI auf und betätigen die Schaltfläche OPTIONEN. Im Dialogfeld EXCEL-OPTIONEN klicken Sie auf die Schaltfläche MENÜBAND ANPASSEN. Auf der rechten Seite markieren Sie die Hauptregisterkarte ENTWICKLERTOOLS (siehe Abbildung 1.13) und betätigen die Schaltfläche OK.

Abbildung 1.13 Registerkarte »Entwicklertools« einblenden

Anschließend ist die Registerkarte dauerhaft aktiviert (siehe Abbildung 1.14).

Abbildung 1.14 Registerkarte »Entwicklertools«

1.2.7 Makrosicherheit dauerhaft ändern

Klicken Sie auf der Registerkarte ENTWICKLERTOOLS die Schaltfläche MAKROSICHERHEIT an (siehe Abbildung 1.15).

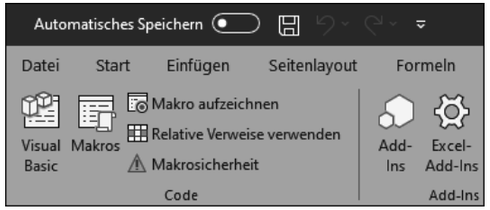


Abbildung 1.15 Schaltfläche »Makrosicherheit«

Anschließend öffnet sich das Dialogfeld TRUST CENTER. Nach Auswahl der Kategorie MAKROEINSTELLUNGEN sehen Sie die Standardeinstellung DEAKTIVIEREN VON VBA-MAKROS MIT BENACHRICHTIGUNG (siehe Abbildung 1.16). Außerdem sehen Sie die verschiedenen Einstellmöglichkeiten.

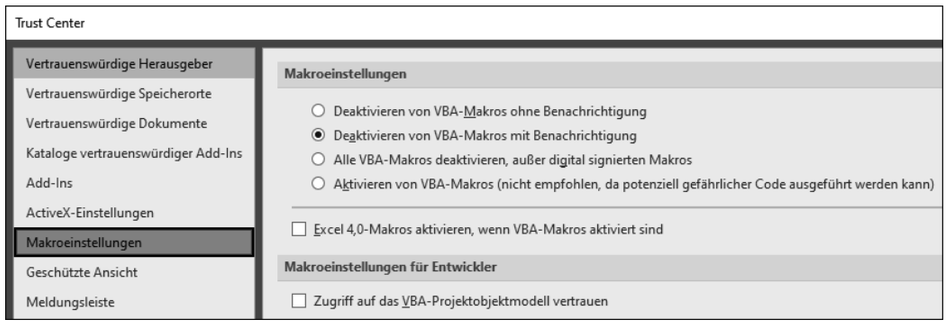


Abbildung 1.16 Trust Center

Ich empfehle Ihnen, die Makroeinstellungen nicht zu ändern. Stattdessen sollten Sie Ihre Excel-Dateien mit Makros in einem bestimmten Verzeichnis speichern und dieses Verzeichnis zu der Kategorie VERTRAUENSWÜRDIGE SPEICHERORTE hinzufügen. Die Makros in den Verzeichnissen dieser Kategorie können immer ausgeführt werden.

Wählen Sie zum Hinzufügen eines Verzeichnisses die Kategorie VERTRAUENSWÜRDIGE SPEICHERORTE aus. Es wird eine Liste von Verzeichnissen angezeigt. Betätigen Sie die Schaltfläche NEUEN SPEICHERORT HINZUFÜGEN. Es erscheint ein Dialogfeld, in dem Sie Ihr Verzeichnis eintragen (siehe Abbildung 1.17).

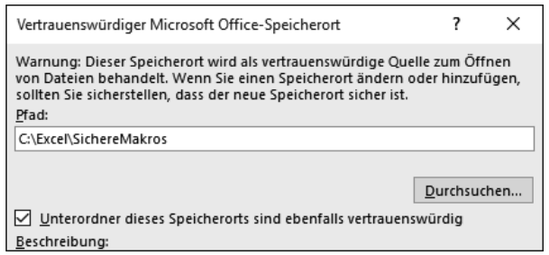


Abbildung 1.17 Vertrauenswürdiges Verzeichnis

Anschließend ist die Liste der Verzeichnisse um dieses Element ergänzt (siehe Abbildung 1.18).

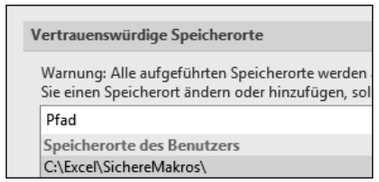


Abbildung 1.18 Liste der vertrauenswürdigen Verzeichnisse

Sind Sie nicht mehr sicher, welche einzelnen Dokumente Sie bereits außerhalb der vertrauenswürdigen Verzeichnisse als vertrauenswürdig erklärt haben, betätigen Sie nach der Auswahl der Kategorie VERTRAUENSWÜRDIGE DOKUMENTE die Schaltfläche LÖSCHEN. Damit wird die gesamte Liste der vertrauenswürdigen Dokumente gelöscht. Anschließend erklären Sie Ihre Dokumente jeweils beim Öffnen wieder einzeln als vertrauenswürdig.

1.3 Visual Basic Editor

Bei dem *Visual Basic Editor* (VBE) handelt es sich um die Entwicklungsumgebung, in der Sie den VBA-Code schreiben. Zum Aufruf des VBE betätigen Sie die Schaltfläche VISUAL BASIC auf der Registerkarte ENTWICKLERTOOLS.

Alternativ rufen Sie den VBE von der Excel-Oberfläche aus mit der Tastenkombination Alt+F11 auf. Mit der gleichen Tastenkombination wechseln Sie auch wieder vom VBE zur Excel-Oberfläche zurück. Dies ist in der Praxis der gängigste Weg.

1.3.1 Menüleiste und Symbolleiste

Im oberen Teil des VBE befinden sich eine Menüleiste und die Symbolleiste VOREINSTELLUNG. Sehr nützlich für die Programmierung ist auch die Symbolleiste BEARBEITEN, die Sie über das Menü ANSICHT • SYMBOLLEISTEN einblenden. Beide Symbolleisten sehen Sie in Abbildung 1.19.

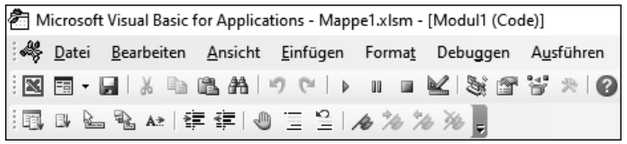


Abbildung 1.19 Symbolleisten »Voreinstellung« und »Bearbeiten«

Nehmen Sie zu Beginn eine wichtige Voreinstellung vor: Im Menü EXTRAS • OPTIONEN • Registerkarte EDITOR setzen Sie ein Häkchen bei VARIABLENDEKLARATION ERFORDERLICH (siehe Abbildung 1.20) und betätigen die Schaltfläche OK.

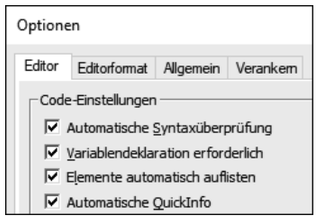


Abbildung 1.20 Einstellung »Variablendeklaration erforderlich«

Dies sorgt ab dem nächsten Öffnen von Excel dafür, dass in jedem Modul oberhalb des VBA-Codes die Zeile `Option Explicit` steht und alle Variablen deklariert werden müssen.

Damit wird in Ihrem VBA-Code automatisch darauf geachtet, dass Sie alle Variablen explizit deklarieren. Dies trägt zur Verminderung von Fehlern und zur Verbesserung der Performance Ihrer Programme bei. Variablen dienen der Speicherung von Zahlen, Daten und Texten, die in einem Programm benötigt werden. Dazu mehr in Kapitel 3, »Grundlagen der Programmierung mit VBA«.

1.3.2 Projekt-Explorer und Fenster »Eigenschaften«

Auf der linken Seite des VBE befinden sich der PROJEKT-EXPLORER und das Fenster EIGENSCHAFTEN (siehe Abbildung 1.21).

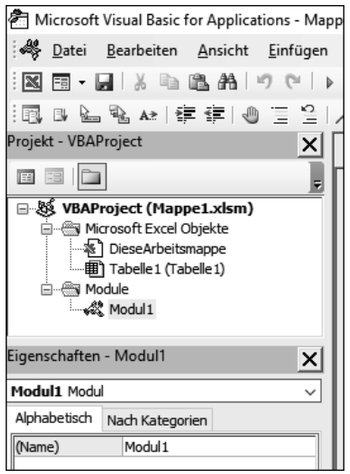


Abbildung 1.21 Projekt-Explorer und Fenster »Eigenschaften«

Im PROJEKT-EXPLORER werden die Elemente der aktuell geöffneten Dateien angezeigt, in denen Sie aktuell mit VBA arbeiten können:

- Es gibt eigene Codemodule, wie z. B. das `Modul1`, das durch die Aufzeichnung eines Makros automatisch erzeugt wurde. Ein eigenes, leeres Modul wird zudem über das Menü EINFÜGEN • MODUL erzeugt.
- Zudem gibt es vorhandene Klassenmodule, wie `DieseArbeitsmappe` oder `Tabelle1`, also Module für die gesamte Arbeitsmappe oder für die aktuell vorhandenen Tabellen. Hier wird VBA-Code notiert, der beim Eintreten eines Ereignisses automatisch ausgeführt wird, das mit einem dieser Objekte zusammenhängt, wie zum Beispiel das Öffnen einer Arbeitsmappe.

- Sie können eigene Klassenmodule zur objektorientierten Programmierung erzeugen, und zwar über das Menü EINFÜGEN • KLASSENMODUL.
- Sie können auch *UserForms*, also eigene Dialogfelder zur komfortablen Programmsteuerung, über das Menü EINFÜGEN • USERFORM erzeugen.

Wir arbeiten zunächst mit eigenen Codemodulen. Die VBA-Beispiele in diesem Buch werden in Codemodulen (*Modul1*, *Modul2* ...) gespeichert, falls kein anderslautender Hinweis erfolgt. Wird Code in *UserForms* oder in eigenen Klassenmodulen gespeichert, so wird dies explizit erwähnt. Per Doppelklick auf ein Element im PROJEKT-EXPLORER wird der VBA-Code des betreffenden Moduls eingeblendet.

> **Übung »Code einblenden«**
>
> Blenden Sie den VBA-Code der verschiedenen Elemente ein.

Im Fenster EIGENSCHAFTEN werden die Eigenschaften des aktuell ausgewählten Moduls bzw. die Eigenschaften des ausgewählten Steuerelements eines eigenen Dialogfelds aufgelistet. Die Werte dieser Eigenschaften lassen sich ändern. Ein Modul wird durch einen einfachen Klick im PROJEKT-EXPLORER ausgewählt. Das Fenster EIGENSCHAFTEN wird für uns erst später wichtig.

1.3.3 Codefenster

Auf der rechten Seite des VBE befindet sich das Codefenster mit dem VBA-Code. Zur direkten Ausführung eines der Makros setzen Sie den Cursor in die betreffende Prozedur, und führen Sie Folgendes aus:

- Rufen Sie im Menü AUSFÜHREN den Menüpunkt SUB/USERFORM AUSFÜHREN auf, oder
- betätigen Sie die Taste F5, oder
- betätigen Sie in der Symbolleiste VOREINSTELLUNG das Symbol mit dem grünen Pfeil, der nach rechts weist (siehe Abbildung 1.22).

Abbildung 1.22 Symbol »Sub/UserForm ausführen« (Taste »F5«)

Übung »Code starten«

Rufen Sie die beiden vorhandenen Makros mit den beschriebenen Methoden auf. Das Ergebnis sehen Sie jeweils auf der Excel-Oberfläche. Sie müssen daher zwischen Excel-Oberfläche und VBE hin- und herwechseln, z. B. mithilfe der Tastenkombination Alt+F11.

Übung »Code kommentieren«

Fügen Sie Ihren beiden Makros jeweils eine Kommentarzeile hinzu mit dem Text `Dieses Makro wurde von [Ihr Name] erstellt.` Zur Erinnerung: Ein Hochkomma im VBA-Code dient dazu, eine ganze Zeile bzw. den Rest einer Zeile hinter dem Hochkomma zu einem Kommentar zu machen. Rufen Sie Ihre Makros erneut auf.

1.4 Makrocode verstehen und ändern

Im nächsten Schritt verändern Sie Ihre eigenen Makros. Dazu betrachten wir den Code etwas näher, zunächst am Beispiel des Makros `Makro1`:

```
Sub Makro1()
    Range("A1").Select
    Selection.Cut
    Range("C1").Select
    ActiveSheet.Paste
End Sub
```

Listing 1.1 Sub »Makro1()« in Mappe1.xlsm, Modul 1

Der Code der Makros wird jeweils innerhalb einer Sub-Prozedur zwischen `Sub` und `End Sub` notiert. Die einzelnen Zeilen werden der Reihe nach durchlaufen und ausgeführt.

Die Anweisung `Range("A1").Select` bedeutet: Für einen Zellbereich (*Bereich*: engl. *range*) wird etwas durchgeführt. In diesem Fall wird der Zellbereich ausgewählt (engl. *select*). Der Zellbereich umfasst hier nur Zelle `A1`. Er kann auch mehrere Zellen umfassen.

Dank der Anweisung `Selection.Cut` wird für die vorher getroffene Auswahl (engl. *selection*) etwas durchgeführt. In diesem Fall wird die Auswahl ausgeschnitten (engl. *cut*), hier der Inhalt von Zelle `A1`. Dieser Inhalt befindet sich nun in der Zwischenablage.

In dem Makro `KopieE3E5` aus der Übung »Makro aufzeichnen« wird auf ähnliche Art und Weise mit `Selection.Copy` die vorher getroffene Auswahl in die Zwischenablage kopiert.

Die Anweisung `Range("C1").Select` erklärt sich nun von selbst.

Die Anweisung `ActiveSheet.Paste` bedeutet: Für das aktuell aktive Tabellenblatt (engl. *sheet*) wird etwas durchgeführt. In diesem Fall wird der Inhalt der Zwischenablage eingefügt (engl. *paste*). Dies wird an der vorher ausgewählten Stelle (Zelle `C1`) vorgenommen.

Hinweis

Das Präfix `Active` bezeichnet immer das aktuell aktive Element, wie z. B. `ActiveCell` (Zelle), `ActiveSheet` (Tabellenblatt) oder `ActiveWorkbook` (Arbeitsmappe).

Ändern Sie den Code durch den Eintrag von `D1` statt `C1` als Zielzelle. Damit wird durch Ausführung des Makros der Inhalt von Zelle `A1` in Zelle `D1` verschoben.

Übung »Makro ändern«

Verändern Sie das Makro `KopieE3E5` aus der Übung »Makro aufzeichnen« (siehe Abschnitt 1.2.1, »Makro aufzeichnen«). Der Inhalt der Zellen `E3` bis `G3` wird sowohl in die Zellen `E5` bis `G5` als auch in die Zellen `E10` bis `G10` kopiert. Nennen Sie das geänderte Makro `KopieE3E5Neu`.

Hinweise dazu:

- Ein Zellbereich aus mehreren Zellen wird genau wie in Excel notiert, also z. B. `E3:G3`.
- Für eine Zielangabe reicht die Angabe einer einzelnen Zelle aus. Die weiteren Zellen des kopierten Bereichs werden, wie aus Excel gewohnt, in die Nachbarzellen eingefügt.
- Aus der Zwischenablage können beliebig viele Kopien eingefügt werden.

Hinweis

In den genannten Beispielen werden Quellbereiche selektiert und Zielbereiche aktiviert. Dadurch erhält man als VBA-Neuling einen leicht verständlichen Einstieg, weil der Code die gleichen Schritte ausführt wie der Benutzer auf der Excel-Oberfläche.

Es wird aber mehr Code als notwendig erzeugt. Die Wartung eines solchen Codes dauert länger, und die Ausführung ist langsamer. Im weiteren Verlauf dieses Buches werden Sie lernen, wie Sie besseren Code erstellen.

1.5 Makro per Schaltfläche ausführen

Stellen Sie der Benutzerin eine Schaltfläche (engl. *button*) zum Starten eines Makros zur Verfügung. Das trägt dazu bei, Benutzern eine übersichtliche Benutzeroberfläche mit Daten, Formeln und Programmen zu bieten.

Zur Erzeugung einer Schaltfläche gehen Sie wie folgt vor:

1. Auf der Excel-Oberfläche betätigen Sie den Pfeil unterhalb der Schaltfläche EINFÜGEN auf der Registerkarte ENTWICKLERTOOLS.
2. Es erscheint eine Sammlung von Steuerelementen. Daraus wählen Sie das Symbol links oben: SCHALTFLÄCHE (FORMULARSTEUERELEMENT).
3. Platzieren Sie den Mauszeiger an der gewünschten Stelle auf dem Tabellenblatt. Es erscheint ein kleines Kreuz. Drücken Sie die linke Maustaste, halten Sie sie gedrückt, und ziehen Sie die Schaltfläche in der gewünschten Größe auf.
4. Das Dialogfeld MAKRO ZUWEISEN (siehe Abbildung 1.23) öffnet sich. Suchen Sie das gewünschte Makro heraus, und bestätigen Sie mit OK.

Abbildung 1.23 Makro einer Schaltfläche zuordnen

Anschließend erscheint die Schaltfläche im Bearbeitungsmodus. Klicken Sie einmal neben die Schaltfläche, damit der Bearbeitungsmodus der Schaltfläche endet. Anschließend können Sie das ausgewählte Makro durch einen Klick auf die Schaltfläche ausführen (siehe Abbildung 1.24).

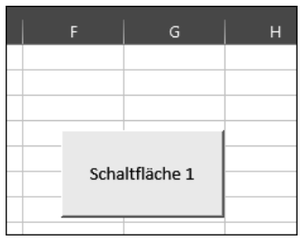


Abbildung 1.24 Neu eingefügte Schaltfläche

Hinweis

Sie können eine Schaltfläche nachträglich bearbeiten. Dazu klicken Sie mit der rechten Maustaste auf die Schaltfläche, dann sind Sie wieder im Bearbeitungsmodus. Nun lassen sich die Größe und die Position der Schaltfläche beeinflussen. Über das Kontextmenü (wiederum erreichbar per Rechtsklick) weisen Sie der Schaltfläche ein anderes Makro zu oder ändern den Text auf der Schaltfläche. Sie können sie gegebenenfalls durch Betätigung der Taste `Entf` löschen.

1.6 Relative Verweise verwenden

Beim Aufzeichnen der Makros ist Ihnen sicherlich schon die Schaltfläche RELATIVE VERWEISE VERWENDEN aufgefallen. Dabei handelt es sich um eine Schaltfläche, die Sie an zwei Stellen betätigen können:

- in dem Menü, das über die Schaltfläche MAKROS auf der Registerkarte ANSICHT erreicht wird
- auf der Registerkarte ENTWICKLERTOOLS

Ist dieser Schalter eingeschaltet, also optisch hervorgehoben, werden die nachfolgenden Makros *relativ* aufgezeichnet. Bei den bisher erstellten Makros war dieser Schalter nicht eingeschaltet. Diese Makros wurden daher *absolut* aufgezeichnet:

- *Absolute Aufzeichnung* bedeutet, dass im Makrocode absolute Zellangaben stehen (Beispiel: `A1`, `C1`). Bei der Ausführung eines solchen Makros ist es egal, welche Zelle in Excel aktiv ist, denn es wird immer mit den Zellen A1 und C1 gearbeitet.
- *Relative Aufzeichnung* bedeutet, dass im Makrocode relative Zellangaben stehen. Bei der Ausführung eines solchen Makros werden die Aktionen relativ zu der Zelle ausgeführt, die in Excel aktiv ist.

Zur Verdeutlichung wird ein Makro aufgezeichnet, in dem das Gleiche gemacht wird wie im ersten Beispielmakro: Der Inhalt von Zelle `A1` wird nach `C1` verschoben. Allerdings wird das Makro relativ aufgezeichnet. Führen Sie folgende Schritte durch:

1. Aktivieren Sie den Schalter RELATIVE VERWEISE VERWENDEN.
2. Tragen Sie in Zelle `A1` einen beliebigen Inhalt ein (Zahl oder Text).
3. Wählen Sie Zelle `A1` aus.
4. Beginnen Sie mit der Aufzeichnung des Makros (Name `Makro2`).
5. Schneiden Sie den Inhalt von Zelle `A1` aus.

6. Fügen Sie den Inhalt der Zwischenablage in Zelle C1 ein. Der Inhalt von Zelle A1 wurde in Zelle C1 verschoben.
7. Beenden Sie die Makroaufzeichnung.

Bis hierhin ist noch kein sichtbarer Unterschied aufgetreten. Führen Sie allerdings das Makro aus, wird jeweils der Inhalt der aktuell ausgewählten Zelle um zwei Zellen nach rechts verschoben, zum Beispiel von Zelle D10 in Zelle F10. Die aktive Zelle ist nun der Ausgangspunkt.

Der Code des neuen Makros:

```
Sub Makro2()
    Selection.Cut
    ActiveCell.Offset(0, 2).Range("A1").Select
    ActiveSheet.Paste
End Sub
```

Listing 1.2 Sub »Makro2()« in Mappe1.xlsm, Modul 1

Betrachten wir den Code:

- `Selection.Cut`: Es geschieht das Gleiche wie bei der absoluten Aufzeichnung: Der Inhalt der aktiven Zelle wird ausgeschnitten. Er liegt nun in der Zwischenablage.
- `ActiveCell.Offset(0,2).Range("A1").Select`: Ausgehend von der aktiven Zelle wird ein *Offset* (also ein Versatz) ausgewählt. Die Angaben in Klammern stehen für Zeile und Spalte des Versatzes. Es wird daher eine Zelle in der gleichen Zeile (Offset-Zeile 0) und zwei Spalten weiter rechts (Offset-Spalte 2) ausgewählt. Ist die aktive Zelle D10, wird demnach Zelle F10 ausgewählt.
- Zu diesem Versatz wird der Zellbereich A1 ausgewählt. Dies ist nicht die absolute Zelle A1 des Tabellenblatts, sondern die oberste linke Zelle des Versatzes. Diese Zelle (in der gleichen Zeile, zwei Zellen weiter rechts) wird ausgewählt.
- `ActiveSheet.Paste`: Es geschieht das Gleiche wie bei der absoluten Aufzeichnung: Der Inhalt der Zwischenablage wird eingefügt.

Dieses Makro verhält sich anders als das Makro, das absolut aufgezeichnet wurde. Beide Aufzeichnungsarten haben ihre Vorteile, je nach Situation.

Hinweis

Nach dem Schließen und erneuten Öffnen von Excel steht der Schalter immer auf »absolute Aufzeichnung«. Während einer Aufzeichnung wechseln Sie bei Bedarf über den Schalter zwischen relativer und absoluter Aufzeichnung.

Übung »Relative Verweise«

Erstellen Sie ein Makro, das den Inhalt von drei nebeneinanderliegenden, markierten Zellen um zwei Zeilen bzw. um sieben Zeilen nach unten kopiert. Es entspricht dem Makro aus der Übung »Makro ändern« (siehe Abschnitt 1.4, »Makrocode verstehen und ändern«), allerdings mit relativer Aufzeichnung. Nennen Sie das Makro `RelKopie`. Testen Sie es, und interpretieren Sie den VBA-Code.

1.7 Persönliche Makroarbeitsmappe

Die bisher aufgezeichneten Makros wurden in der aktuell aktiven Arbeitsmappe gespeichert. Daher stehen sie nur dort zur Verfügung. Möchten Sie bestimmte Makros jederzeit zur Verfügung haben, speichern Sie sie in der Mappe *Persönliche Makroarbeitsmappe*. Diese Arbeitsmappe wird immer zusammen mit Excel geöffnet, sobald sie existiert.

Zum Speichern eines Makros in der Mappe *Persönliche Makroarbeitsmappe* wird im Dialogfeld MAKRO AUFZEICHNEN der entsprechende Eintrag in der Liste MAKRO SPEICHERN IN: ausgewählt (siehe Abbildung 1.25).

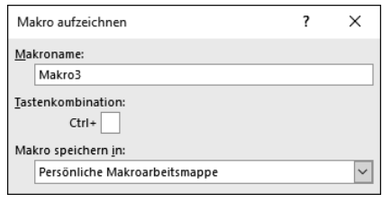


Abbildung 1.25 In der Mappe »Persönliche Makroarbeitsmappe« speichern

Nach der Aufzeichnung erscheint ein weiterer Eintrag im PROJEKT-EXPLORER des VBE: die Datei *personal.xlsb* (siehe Abbildung 1.26).

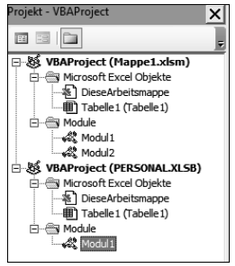


Abbildung 1.26 »Persönliche Makroarbeitsmappe« im Projekt-Explorer

Wurde diese Datei verändert, indem z. B. ein Makro hinzugefügt oder geändert wurde, werden Sie bei jedem Schließen von Excel gefragt, ob Sie diese Änderungen speichern möchten.

> **Übung »Persönliche Makroarbeitsmappe«**
>
> Erstellen Sie noch einmal das Makro aus der Übung »Relative Verweise«. Speichern Sie es diesmal in der Mappe *Persönliche Makroarbeitsmappe*. Schließen Sie Excel. Testen Sie das Makro nach dem erneuten Öffnen von Excel in einer beliebigen Arbeitsmappe.

1.8 Code schreiben für einfache Ausgaben

In diesem Abschnitt wird erstmalig VBA-Code nicht durch eine Makroaufzeichnung, sondern durch Schreiben des Codes in einer eigenen Sub-Prozedur erstellt. Damit entsteht das erste richtige VBA-Programm. ich werde drei Möglichkeiten zur Ausgabe von Ergebnissen oder Kontrollwerten erläutern, die häufig in der VBA-Programmierung eingesetzt werden:

- in einer Zelle
- in einer Nachrichtenbox
- im Direktfenster des VBE

1.8.1 Eigene Sub-Prozedur

Zunächst erstellen Sie eine eigene Sub-Prozedur. Wechseln Sie dazu mit der Tastenkombination Alt+F11 zum VBE. Schreiben Sie unterhalb des letzten Makros die folgende Zeile:

```
Sub AusgabeZelle
```

Sobald Sie mit der Taste ↵ die Zeile wechseln, stellen Sie fest, dass der Editor Sie bei der VBA-Programmierung unterstützt (siehe Abbildung 1.27):

- Am Ende der Zeile werden Klammern angefügt.
- Es wird die Zeile mit `End Sub` erzeugt.
- Der Cursor steht in der Zeile dazwischen – bereit zur Eingabe Ihres VBA-Codes.
- Es wird eine Trennzeile zwischen den einzelnen Makros erzeugt.


```
End Sub

Sub AusgabeZelle()

End Sub
```

Abbildung 1.27 Neue eigene Sub-Prozedur

Nach dem Namen einer Sub-Prozedur (oder kurz Prozedur) folgen immer Klammern, wie bei Ihren bisherigen Makros. Diese Klammern sind zunächst leer. Später lernen Sie, innerhalb der Klammern Aufrufparameter zu notieren, die eine Prozedur flexibler machen.

1.8.2 Ausgabe in Zelle

Geben Sie in der Prozedur `AusgabeZelle()` folgende Codezeile ein:

```
Range("A1").Value = "Hallo"
```

Damit sieht Ihre Prozedur wie folgt aus:

```
Sub AusgabeZelle()
    Range("A1").Value = "Hallo"
End Sub
```

Listing 1.3 Sub »AusgabeZelle()« in Mappe1.xlsm, Modul 1

Beim Ausführen dieser Prozedur, z. B. mithilfe der Taste F5, wird der Text »Hallo« in Zelle `A1` ausgegeben (siehe Abbildung 1.28).

	A	B
1	Hallo	
2		

Abbildung 1.28 Ergebnis der eigenen Sub-Prozedur

Auf einen Zellbereich, der eine oder mehrere Zellen umfasst, wird über `Range` zugegriffen. Der Eigenschaft `Value` dieses Bereichs wird ein Wert zugewiesen, der Text »Hallo«. Damit erhält die Zelle einen Inhalt.

> **Hinweis**
>
> Geben Sie einer Prozedur einen eindeutigen und sprechenden Namen, der etwas über ihre Arbeitsweise aussagt.

1.8.3 Ausgabe in Nachrichtenbox

Schreiben Sie eine weitere Prozedur mit folgendem Code:

```
Sub AusgabeBox()
    MsgBox "Hallo"
End Sub
```

Listing 1.4 Sub »AusgabeBox()« in Mappe1.xlsm, Modul 1

Nach dem Aufruf der Prozedur erscheint eine kleine Nachrichtenbox mit dem Text »Hallo« (siehe Abbildung 1.29). Das Programm hält an. Nach dem Betätigen der Schaltfläche OK wird die Nachrichtenbox geschlossen, und das Programm läuft weiter.

Zeigt der VBE einmal unerwartet keinerlei Reaktion, wechseln Sie mit der Tastenkombination Alt+F11 zum Excel-Tabellenblatt: Möglicherweise wartet dort eine Nachricht auf Ihre Reaktion.

Abbildung 1.29 In Nachrichtenbox ausgeben

Eine solche Nachrichtenbox benutzen Sie u. a. zur:

- Kontrollausgabe während der Programmierung
- Information der Benutzerin bei einem Fehler
- Information des Benutzers am Ende eines Programms

Bei `MsgBox()` handelt es sich um eine vorgefertigte Funktion, bei der Sie eine Zeichenkette als Aufrufparameter notieren müssen. Die Funktion `MsgBox()` bietet allerdings wesentlich mehr, wie Sie in Abschnitt 8.3, »Einfacher Dialog mit dem Benutzer«, sehen werden.

Bei längeren Ausgaben wird häufig ein Zeilenumbruch benötigt. Dieser wird über die Konstante `vbCrLf` bereitgestellt. Ein Beispiel:


```
Sub AusgabeBoxZeilenumbruch()
    MsgBox "Hallo" & vbCrLf & "Welt"
End Sub
```

Listing 1.5 Sub »AusgabeBoxZeilenumbruch()« in Mappe1.xlsm, Modul 1

Die Ausgabe sehen Sie in Abbildung 1.30.

Abbildung 1.30 Mit Zeilenumbruch ausgeben

Bei dem Namen dieser Konstanten handelt es sich um ein Relikt aus der Urzeit: In `vbCrLf` steht das `Cr` für *Carriage Return*, den Wagenrücklauf der Schreibmaschine zum Zeilenanfang. Das `Lf` steht für *Line Feed*, den Zeilenvorschub.

> **Hinweis**
>
> Der Operator `&` dient der Verkettung von Zeichenfolgen. Er wird häufig benötigt, um die Ergebnisse eines Programms zusammen mit Text auszugeben. Dabei werden Zahlen vor der Verkettung automatisch in Text umgewandelt.

1.8.4 Ausgabe im Direktfenster des VBE

Schreiben Sie eine weitere Prozedur mit folgendem Code:

```
Sub AusgabeKontrolle()
    Debug.Print "Hallo"
End Sub
```

Listing 1.6 Sub »AusgabeKontrolle()« in Mappe1.xlsm, Modul 1

Nach dem Aufruf erscheint zunächst gar nichts, da das Direktfenster des VBE standardmäßig nicht eingeblendet ist. Lassen Sie es über das Menü Ansicht • Direktfenster im VBE anzeigen. Anschließend sehen Sie die soeben erzeugte Ausgabe (siehe Abbildung 1.31).

Das Direktfenster (= Direktbereich) nutzen Sie ebenfalls zur Kontrollausgabe während der Programmierung.

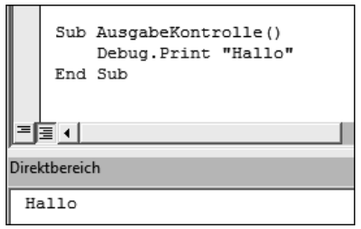


Abbildung 1.31 Im Direktfenster ausgeben

1.9 Microsoft 365

Bei *Microsoft 365* handelt es sich um ein Software-Abonnement, das u. a. zur Nutzung von Excel dient. Funktionen in neuen Excel-Versionen, wie zum Beispiel die Überlauf-Funktionen für Excel 2021 (siehe Kapitel 11, »Funktionen aus Excel 2021«), stehen auch in Microsoft 365 zur Verfügung. Zudem haben Sie Zugang zu folgenden Möglichkeiten:

- *Microsoft OneDrive*: ein Dateispeicherdienst zum Speichern Ihrer Excel-Dateien in der Microsoft Cloud
- *Microsoft Office Online*: eine Online-Anwendung zum Bearbeiten Ihrer Excel-Dateien in einem Browser

Ihren Zugang erhalten Sie mit Ihren persönlichen Anmeldedaten zu Microsoft über die Adresse *https://onedrive.live.com/*.

Sie speichern Excel-Dateien mit Makros, die Sie mit der Desktop-Anwendung auf Ihrem PC erstellt haben, entweder auf Ihrem PC oder in der Microsoft Cloud. Im letztgenannten Fall werden die Dateien zum Beispiel unter der folgenden Adresse gespeichert:

https://d.docs.live.net//xxx/Documents/Mappe1.xlsm

Die Zeichenfolge *xxx* steht hier für Ihren persönlichen Zugang bei Microsoft. Dateien, die Sie mithilfe von Microsoft OneDrive in der Microsoft Cloud gespeichert haben, können Sie auch wieder auf Ihren PC laden.

Öffnen Sie eine Excel-Datei, die VBA-Code enthält und in der Microsoft Cloud gespeichert ist, müssen Sie die Makros zunächst aktivieren, siehe auch Abschnitt 1.2.5, »Makrosicherheit ändern«. Anschließend haben Sie die Möglichkeit, diese Datei zu einem vertrauenswürdigen Dokument zu erklären, siehe Abbildung 1.32. Entscheiden Sie sich für diese Möglichkeit, werden Sie beim nächsten Öffnen der Datei nicht mehr gefragt.

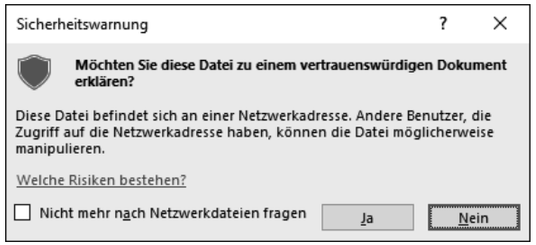


Abbildung 1.32 Datei mit Netzwerkadresse

Die Online-Anwendung von Excel in Microsoft Office Online hat gegenüber der Desktop-Anwendung einen reduzierten Funktionsumfang. Die Ausführung von Makros beziehungsweise VBA-Code ist nicht möglich. Steuerelemente, zum Beispiel Schaltflächen, werden in der Online-Anwendung nicht angezeigt. Diese Makros und Steuerelemente werden aber nicht gelöscht. Nach dem Öffnen der betreffenden Excel-Datei in der Desktop-Anwendung auf Ihrem PC können Sie sie wieder nutzen.

1.10 Web-Apps

MS Office-Dateien können mit Web-Apps aus dem Office Store erweitert werden. Diese Web-Apps haben Zugriff auf die Daten in einer Office-Datei. Entwickler können diese Web-Apps mithilfe von Webtechniken (HTML, JavaScript, jQuery, CSS ...) programmieren und im Office Store anbieten. In diesem Buch wird nur die Entwicklung von Anwendungen mithilfe von VBA beschrieben.

Kapitel 3
Grundlagen der Programmierung mit VBA

Inhalte dieses Kapitels sind Variablen, Datentypen, Operatoren und Kontrollstrukturen im Zusammenspiel mit Excel und seinen Objekten.

Im Folgenden lernen Sie die wichtigsten grundlegenden Elemente und Techniken kennen, die Sie zum Schreiben von VBA-Programmen benötigen.

3.1 Allgemeines

Vor der Programmierung folgen noch einige allgemeine Tipps.

Eine Entwicklerin, ob in VBA oder in einer anderen Programmiersprache, sollte sich um einen guten Programmierstil bemühen. Sie erleichtert sich und anderen damit die Arbeit. Durch eine übersichtliche Schreibweise und eine ausreichende Kommentierung ist der Code von ihr und von anderen besser lesbar und wartbar.

Zu einer übersichtlichen Schreibweise gehören z. B. Einrückungen und Leerzeilen. Die Entwicklungsumgebung VBE unterstützt Sie als Entwicklerin mit zahlreichen Hilfestellungen.

Einige dieser Hilfestellungen kennen Sie vermutlich schon:

- Markieren Sie den Namen eines Objekts oder einer Eigenschaft und betätigen Sie anschließend die Taste F1, wird die zugehörige Erläuterung eingeblendet. Kann der Begriff nicht eindeutig zugeordnet werden, wird eine Liste mit den möglichen Zuordnungen angezeigt, aus der Sie auswählen.
- Werden Sie nicht zur richtigen Seite der VBA-Hilfe geleitet oder finden Sie keinen passenden Hinweis für den Weg dorthin, geben Sie einfach den Begriff »VBA« und den Suchbegriff in einer Suchmaschine ein. Meist landen Sie dann schnell auf der richtigen Seite des *Microsoft Developer Network* (MSDN) für VBA.
- Sobald Sie eine Zeile beendet haben und zur nächsten Zeile wechseln, werden Korrekturen vorgenommen. Ein Beispiel:

  - Sie schreiben »`msgbox activesheet.name`«, eventuell sogar mit mehreren unnötigen Leerzeichen hinter `msgbox`.
  - Im Editor wird dies beim Zeilenwechsel geändert in `MsgBox ActiveSheet.Name`. Alle Leerzeichenfolgen werden außerdem reduziert zu einem Leerzeichen.
- Bei einem Wechsel von einer eingerückten Zeile zur nächsten Zeile bleibt die Einrückung erhalten. Dadurch sind Programmstrukturen, wie Prozeduren, Verzweigungen und Schleifen, leichter erkennbar.
- Sie rücken mehrere Zeilen auf einmal ein, indem Sie sie vollständig markieren und dann die Taste ⇥ betätigen. Mit der Tastenkombination ⇧+⇥ machen Sie Einrückungen wieder rückgängig.
- Sobald Sie einen Punkt hinter den Namen eines Objekts gesetzt haben, erscheint eine Liste der Eigenschaften und Methoden dieses Objekts zur Auswahl. Geben Sie einen oder mehrere Anfangsbuchstaben ein, wird das passende Element in der Liste ausgewählt. Mit der Taste ⇥ fügen Sie das betreffende Listenelement in den Code ein.

3.1.1 Codeblöcke auskommentieren

Sie können ganze Codeblöcke als Kommentar setzen, damit der betreffende Code kurzfristig nicht durchlaufen wird. Dies können z. B. Zeilen mit Kontrollausgaben (`MsgBox`, `Debug.Print`) sein, die in der Endfassung des Programms nicht mehr benötigt werden.

Zur Auskommentierung markieren Sie die betreffenden Zeilen. Anschließend betätigen Sie das Symbol Block auskommentieren in der Symbolleiste Bearbeiten (siehe Abbildung 3.1). Rechts daneben steht das Symbol Auskommentierung des Blocks aufheben, mit dem Sie den VBA-Code wieder aktivieren. Die Symbolleiste Bearbeiten lässt sich über das Menü Ansicht • Symbolleisten einblenden.

Abbildung 3.1 Block auskommentieren und Auskommentierung wieder aufheben

> **Übung »Code auskommentieren«**
>
> Verändern Sie das Makro `KopieE3E5Neu` aus der Übung »Makro ändern«, siehe Abschnitt 1.4, »Makrocode verstehen und ändern«. Kommentieren Sie einzelne Zeilen aus, so dass die drei Zellen `E3` bis `G3` nur noch in die Zellen `E10` bis `G10` kopiert werden, aber nicht mehr in die Zellen `E5` bis `G5`. Geben Sie am Ende den Text »Fertig« in einer Nachrichtenbox aus. Testen Sie das veränderte Makro.

3.1.2 Zeilen zerlegen

Lange Programmzeilen lassen sich mithilfe des Zeichens _ (Unterstrich) in mehrere, kürzere Programmzeilen zerlegen. Dies wird hier zum Kopieren eines Tabellenblatts durchgeführt (siehe Abschnitt 2.3.2, »Tabellenblatt kopieren«) – zugegeben in etwas übertriebener Form:

```
Sub BlattKopieren()
    ThisWorkbook.Activate

    Worksheets _
        ("Tabelle1"). _
        Copy _
        After:=Worksheets("Tabelle1")

    ActiveSheet.Name = "Tab1Kopie"
End Sub
```

Listing 3.1 Sub »BlattKopieren()« in Mappe2.xlsm, Modul 2 (geändert)

Am Ende einiger Zeilen steht ein Unterstrich. Dies bedeutet, dass die Anweisung in der nächsten Zeile fortgesetzt wird. Eine übliche Trennstelle ist z. B. ein Leerzeichen, wie hier zwischen `Copy` und `After`. Eine Trennstelle darf nicht innerhalb eines Worts oder innerhalb einer Zeichenkette liegen.

Hier werden als weitere Trennstellen das Ende des Worts `Worksheets` und der Punkt vor `Copy` genutzt. In VBA kommen bei der Referenzierung von Objekten (siehe Abschnitt 6.2) bisweilen sehr lange Objektnamen vor. Das Beispiel zeigt, wie ein solcher Name zerlegt wird.

3.2 Variablen und Datentypen

Variablen dienen der vorübergehenden Speicherung von Daten, die sich zur Laufzeit eines Programms ändern können.

3.2.1 Namen, Werte

Eine Variable besitzt einen eindeutigen Namen, unter dem sie angesprochen wird. Für diese Namen gelten in VBA die folgenden Regeln:

- Sie beginnen mit einem Buchstaben.
- Sie bestehen nur aus Buchstaben, Zahlen und einigen wenigen Sonderzeichen (z. B. dem Unterstrich _).
- Sie enthalten keine deutschen Umlaute (ä, ö, ü) oder das Eszett (ß).

In einem Gültigkeitsbereich darf es keine zwei Variablen mit dem gleichen Namen geben (siehe Abschnitt 5.1, »Gültigkeitsbereiche«).

Gültige Namen sind demnach `Temperatur`, `Summe_Werte` oder `X12`. Ungültige Namen sind z. B. `5Tage` oder `Tag#12`.

Variablen erhalten ihre Werte durch Zuweisung per Gleichheitszeichen. Einer Variablen sollte vor ihrer ersten Benutzung ein Wert zugewiesen werden, z. B. `Temperatur = 25`. Dadurch werden Programme lesbarer und fehlerfreier.

3.2.2 Deklarationen

Variablen sollten in Visual Basic immer deklariert werden. Dies beugt Fehlern und unnötig hohem Speicherbedarf vor. Bei einer Deklaration geben Sie den Namen und den Datentyp einer Variablen an.

Der Datentyp einer Variablen bestimmt die Art der gespeicherten Information. Der Entwickler wählt den Datentyp danach aus, ob er Text, Zahlen ohne Nachkommastellen, Zahlen mit Nachkommastellen oder z. B. Datumsangaben speichern möchte.

Außerdem macht er sich Gedanken über die Größe des Zahlenbereichs, in dem eine Zahl liegt, und verwendet anschließend den Datentyp mit dem geringsten Speicherbedarf. In Abschnitt 3.2.3, »Datentypen«, finden Sie eine Liste der Datentypen mit ihrem Namen, ihrem Speicherbedarf und ihrem Wertebereich.

In Kapitel 1, »Einführung«, haben Sie eine Voreinstellung vorgenommen, die für die VBA-Programmierung wichtig ist: Im Menü Extras • Optionen haben Sie auf der Registerkarte Editor ein Häkchen bei Variablendeklaration erforderlich gesetzt. Dies sorgt ab dem nächsten Öffnen von Excel dafür, dass oberhalb des VBA-Codes die Zeile `Option Explicit` steht und alle Variablen deklariert werden müssen.

3.2.3 Datentypen

Tabelle 3.1 enthält die wichtigsten von VBA unterstützten Datentypen sowie deren Speicherbedarf und Wertebereich:

Datentyp	Speicherbedarf	Wertebereich, Bedeutung
`Boolean`	2 Byte	`True` oder `False` (wahr oder falsch)
`Byte`	1 Byte	ganze Zahl von 0 bis 255
`Integer`	2 Byte	ganze Zahl von –32.768 bis +32.767
`Long`	4 Byte	lange ganze Zahl von $-2{,}1 \times 10^9$ bis $+2{,}1 \times 10^9$
`Single`	4 Byte	Gleitkommazahl mit einfacher Genauigkeit: von ca. $-3{,}4 \times 10^{+38}$ bis ca. $-1{,}4 \times 10^{-45}$ für negative Werte und von ca. $+1{,}4 \times 10^{-45}$ bis ca. $+3{,}4 \times 10^{+38}$ für positive Werte
`Double`	8 Byte	Gleitkommazahl mit doppelter Genauigkeit: von ca. $-1{,}8 \times 10^{+308}$ bis ca. $-4{,}9 \times 10^{-324}$ für negative Werte und von ca. $+4{,}9 \times 10^{-324}$ bis $+1{,}9 \times 10^{+308}$ für positive Werte
`Date`	8 Byte	Datum vom 1. Januar 100 bis zum 31. Dezember 9999
`Object`	4 Byte	Verweis auf ein Objekt, siehe Abschnitt 6.4, »Mit Objektvariablen arbeiten«
`String`	10 Byte (+)	10 Byte plus die Zeichenfolgenlänge, d. h. Zeichenkette mit variabler Länge
`Variant`	>= 16 Byte	Datentyp nicht explizit festgelegt (nicht zu empfehlen)

Tabelle 3.1 Datentypen

Mit folgender Prozedur werden Variablen einiger der oben genannten Typen deklariert, mit Werten versehen und in Zellen angezeigt:

```
Sub Variablen()
    Dim By As Byte
    Dim Bo As Boolean
    Dim It As Integer, Lg As Long
    Dim Sg As Single, Db As Double
    Dim Dt As Date
    Dim St As String

    By = 200
    Bo = True
    It = 20000
    Lg = 200000
```

```
    Sg = 0.1 / 7
    Db = 0.1 / 7
    Dt = "15.08.2022"
    St = "Zeichenkette"

    ThisWorkbook.Worksheets("Tabelle1").Activate

    Range("A1").Value = By
    Range("A2").Value = Bo
    Range("A3").Value = It
    Range("A4").Value = Lg

    Range("A5").Value = Sg
    Range("A6").Value = Db
    Range("A5:A6").NumberFormatLocal = "0,00000000000000000000"

    Range("A7").Value = Dt
    Range("A8").Value = St

    Range("A:A").Columns.AutoFit
End Sub
```

Listing 3.2 Sub »Variablen()« in Mappe3.xlsm, Modul 1

Abbildung 3.2 zeigt das Ergebnis.

	A
1	200
2	WAHR
3	20000
4	200000
5	0,01428571436554190000
6	0,01428571428571430000
7	15.08.2022
8	Zeichenkette
9	

Abbildung 3.2 Verschiedene Datentypen

Variablen werden mit `Dim` deklariert. Mit `As` wird ihnen ein definierter Datentyp zugewiesen. Es können auch mehrere Variablen in einer Zeile deklariert werden. Beachten Sie dabei: Mit `Dim a, b As Integer` wird aus `a` eine Variable vom Datentyp `Variant`! Richtig heißt es: `Dim a As Integer, b As Integer`.

Logische Variablen des Datentyps `Boolean` können nur einen der beiden Wahrheitswerte `True` (wahr) oder `False` (falsch) annehmen.

Bei den Datentypen für Zahlen führt eine Über- oder Unterschreitung des Wertebereichs zu einem Abbruch.

Ganze Zahlen werden in Variablen der Datentypen `Byte`, `Integer` oder `Long` mathematisch genau abgespeichert. Zahlen mit Nachkommastellen werden in Variablen der Datentypen `Single` oder `Double` mit begrenzter Genauigkeit abgespeichert. Die Zahl 18,55 kann z. B. als 18,549999 abgespeichert werden. In vielen Berechnungen ist diese kleine Ungenauigkeit nicht von Belang. Behalten Sie sie aber im Hinterkopf.

Werte für Zahlen mit Nachkommastellen werden im VBA-Code mit einem Dezimalpunkt eingegeben. Die Datentypen `Single` und `Double` für solche Zahlen unterscheiden sich in ihrer Genauigkeit. Im Beispiel wurde die Anzahl der in den Zellen angezeigten Nachkommastellen mithilfe der Eigenschaft `NumberFormatLocal` auf den Wert 20 gestellt. Der Wert einer `Double`-Variablen ist mathematisch genauer als der Wert der `Single`-Variablen.

Hinweis

Wird einer Variablen kein definierter Datentyp (mit `As`) zugewiesen, hat diese Variable den Typ `Variant`. Verwenden Sie diesen Datentyp möglichst selten, da Variablen dieses Datentyps einen höheren Speicherbedarf haben, langsamer zu verarbeiten und schwerer zu kontrollieren sind.

Markieren Sie den Namen einer Variablen. Mit ⇧+F2 gelangen Sie jetzt zu der Stelle, an der die Variable deklariert wurde. Mit Strg+⇧+F2 kommen Sie wieder zurück in die Zeile, in der die Variable genutzt wird.

Bei der Eingabe von sehr großen oder sehr kleinen Werten können Sie mit Exponentialzahlen arbeiten. Zwei Beispiele:

- Die Eingabe `1.5e6` entspricht mathematisch dem Wert $1{,}5 \times 10^6$ und wird vom Editor in den Wert `1500000` umgewandelt.
- Die Eingabe `1.5e-6` entspricht mathematisch dem Wert $1{,}5 \times 10^{-6}$ und wird vom Editor in den Wert `0.0000015` umgewandelt.

Werte für Zeichenketten und Datumsangaben werden in doppelten Anführungszeichen angegeben. Bei Datumsangaben ist die Schreibweise `TT.MM.JJJJ` zu bevorzugen.

Zeilenhöhe und Spaltenbreite lassen sich über die Methode `AutoFit()` des Objekts `Columns` bzw. `Rows` optimal einstellen, siehe auch Abschnitt 2.4.15, »Zeilenhöhe und Spaltenbreite«.

Den selten genutzten Datentyp für *Zeichenketten mit fester Länge* werde ich in Abschnitt 9.4, »Dateien mit wahlfreiem Zugriff«, an einem passenden Beispiel erläutern.

> **Übung »Variablen und Datentypen«**
>
> Schreiben Sie die Prozedur `UebungVariablen()`, in der Ihr Nachname, Ihr Vorname, Ihre Adresse, Ihr Geburtsdatum und Ihr Alter jeweils in Variablen eines geeigneten Datentyps gespeichert und anschließend wie in Abbildung 3.3 ausgegeben werden.

Max Mustermann
Holzweg 7, 53776 Rheinhausen
geb.: 20.07.1978
Alter: 44

Abbildung 3.3 Ausgabe zu Übung »Variablen und Datentypen«

3.2.4 Funktion »TypeName()«

Die Funktion `TypeName()` liefert den Datentyp einer Variablen, eines Zellinhalts oder eines Objekts als Zeichenkette. Ein Beispiel:

```
Sub TypErkennen()
    ThisWorkbook.Worksheets("Tabelle1").Activate
    Range("B2").Value = TypeName(Range("A2").Value)
    Range("B3").Value = TypeName(Range("A3").Value)
    Range("B4").Value = TypeName(Range("A4").Value)
    Range("B5").Value = TypeName(Range("A5").Value)
    Range("B6").Value = TypeName(Range("A6").Value)
    Range("B7").Value = TypeName(Range("A7").Value)
    Range("B8").Value = TypeName(Range("A8").Value)
End Sub
```

Listing 3.3 Sub »TypErkennen()« in Mappe3.xlsm, Modul 1

In den Zellen `B1` bis `B8` wird jeweils der Datentyp der linken Nachbarzelle ausgegeben, also der Zellen `A1` bis `A8`.

Das Ergebnis sehen Sie in Abbildung 3.4.

	A	B
1	200	Double
2	WAHR	Boolean
3	20000	Double
4	200000	Double
5	0,01428571436554190000	Double
6	0,01428571428571430000	Double
7	15.08.2022	Date
8	Zeichenkette	String
9		

Abbildung 3.4 Datentypen

Die Zahlen in den Zellen einer Tabelle werden, ob mit oder ohne Nachkommastellen, als Werte vom Datentyp `Double` erkannt. Ein Beispiel dazu: Die Zahl in Zelle `A5` »weiß nicht mehr«, dass sie ursprünglich einmal eine `Single`-Variable in einem VBA-Programm war. Es könnte ebenso eine Zahl sein, die der Benutzer von Hand in die Zelle eingetragen hat. Die Funktion `TypeName()` des VBA-Programms »sieht« nur, dass es sich um eine Zahl handelt, und liefert den Datentyp mit der höchsten Genauigkeit für Zahlen, also `Double`.

3.2.5 Konstanten

Konstanten kommen seltener zum Einsatz als Variablen. Bei Konstanten handelt es sich um vordefinierte Werte verschiedener Datentypen, die während der Laufzeit nicht verändert werden können. Man gibt Konstanten im Allgemeinen aussagekräftige Namen. Dadurch sind sie leichter zu behalten als die Werte, die sie repräsentieren.

Das Programm greift schneller auf Konstanten als auf Variablen zu. Daher sollten Sie Konstanten verwenden, falls der Wert feststeht und sich während des Programmablaufs niemals ändert. Es gibt:

- *Eigene Konstanten*: Sie werden von der Entwicklerin an einer zentralen Stelle definiert und an verschiedenen Stellen des Programms genutzt. Der Gültigkeitsbereich von Konstanten ist analog zum Gültigkeitsbereich von Variablen (siehe Abschnitt 5.1, »Gültigkeitsbereiche«).
- *Integrierte Konstanten*: Sie sind vordefiniert und können vom Entwickler nicht verändert werden:
  - Beim Einfügen von Zellen verwenden Sie die integrierten Konstanten `xlShiftDown` und `xlShiftToRight`. Sie repräsentieren die Zahlen 4.121 bzw. 4.161.
  - Eine andere häufig genutzte Konstante ist `vbCrLf` für den Zeilenumbruch in einer Nachrichtenbox (`MsgBox`).

Im folgenden Beispiel werden Konstanten und Variablen genutzt:

```
Sub Konstanten()
    Const MaxWert As Integer = 55
    Dim MinWert As Integer
    MinWert = 15
    MsgBox MaxWert - MinWert
    MinWert = 35
    MsgBox MaxWert - MinWert
End Sub
```

Listing 3.4 Sub »Konstanten()« in Mappe3.xlsm, Modul 1

Die Konstante `MaxWert` mit dem Datentyp `Integer` wird festgelegt. Sie kann nicht mehr verändert werden. Dagegen kann die Variable `MinWert` mit dem Datentyp `Integer` in der Prozedur ihren Wert verändern.

3.2.6 Enumerationen

Enumerationen sind Sammlungen von zusammengehörigen, ganzzahligen Konstanten, die ebenfalls aussagekräftige Namen haben. Als Entwickler können Sie sowohl eigene Enumerationen definieren als auch vordefinierte Enumerationen einsetzen.

Im folgenden Beispiel wird eine eigene Enumeration definiert und genutzt.

```
Enum Farbe
    Rot
    Gelb
    Blau
    Schwarz = 5
    Orange
End Enum

Sub Enumeration()
    Dim F As Farbe
    F = Orange
    MsgBox F
End Sub
```

Listing 3.5 Definition und Sub-Prozedur in Mappe3.xlsm, Modul 1

Enumerationen werden innerhalb eines Blocks zwischen `Enum` und `End Enum` definiert, und zwar außerhalb von Sub-Prozeduren, im Kopf des Moduls direkt unterhalb von `Option Explicit`.

Im Beispiel sehen Sie die Enumeration `Farbe` mit fünf Elementen:

- Werden bei der Definition keine Werte zugewiesen, repräsentiert das erste Element den Wert 0, die anschließenden Elemente repräsentieren die nachfolgenden Zahlen: 1, 2, 3 ...
- Wird bei der Definition ein Wert zugewiesen, repräsentiert das betreffende Element diesen Wert (hier: 5), die anschließenden Elemente repräsentieren die nachfolgenden Zahlen (hier: 6, 7, 8 ...).

In der Prozedur `Enumeration()` wird eine Variable des Enumerationstyps `Farbe` deklariert. Bei der Zuweisung eines Wertes erscheint nach Eingabe des Gleichheitszeichens als Hilfestellung eine Liste der Enumerationswerte. Es dürfen zwar auch andere Werte zugewiesen werden, dies würde aber dem Sinn einer Enumeration widersprechen. Als Ausgabe erscheint der Zahlenwert, den das Element der Enumeration repräsentiert (hier: 6).

Es gibt viele vordefinierte Enumerationen. Ein Beispiel aus Abschnitt 2.4.8, »Zellen ausrichten«: Die Eigenschaft `Weight` bestimmt die Stärke des Rahmens eines Zellbereichs. Zulässige Konstanten aus der Enumeration `xlBorderWeight` sind `xlHairline` (ganz feine Linie), `xlThin` (feine Linie), `xlMedium` (mittelstarke Linie) und `xlThick` (starke Linie).

3.3 Operatoren

Zum Zusammensetzen von Ausdrücken werden sowohl in den Zellen der Excel-Tabellenblätter als auch in VBA Operatoren verwendet. Sie haben schon den Operator `=` für die Zuweisung von Werten kennengelernt. Es gibt verschiedene Kategorien von Operatoren. Manche Operatoren arbeiten anders als in Excel.

Werden mehrere Operatoren in einem Ausdruck verwendet, bestimmt die Rangfolge der Operatoren (Priorität) die Reihenfolge bei der Bearbeitung. Diese Rangfolge finden Sie in Abschnitt 3.3.5, »Rangfolge der Operatoren«. Sind Sie sich bei der Verwendung dieser Regeln nicht sicher, können Sie die Reihenfolge der Bearbeitung durch Klammern festlegen.

3.3.1 Arithmetische Operatoren

Mit arithmetischen Operatoren werden Berechnungen durchgeführt. Tabelle 3.2 listet die arithmetischen Operatoren auf.

Operator	Beschreibung
+	Addition
-	Subtraktion oder Negation
*	Multiplikation
/	Division
\	Ganzzahldivision
Mod	Modulo
^	Potenzierung

Tabelle 3.2 Arithmetische Operatoren

Division

Beachten Sie bei der mathematischen Division mit dem Operator / den Datentyp der Variablen, in der das Ergebnis gespeichert wird. In den beiden nachfolgenden Prozeduren wird zunächst auf der rechten Seite des Gleichheitszeichens der mathematische Wert 2,6 berechnet. Dieser Wert wird in der ersten Prozedur einer Double-Variablen zugewiesen, die anschließend den Wert 2,6 besitzt..

Erstes Beispiel:

```
Sub DivisionOhneRunden()
    Dim x As Double
    x = 13 / 5
    MsgBox x
End Sub
```

Listing 3.6 Sub »DivisionOhneRunden()« in Mappe3.xlsm, Modul 1

In der nachfolgenden zweiten Prozedur wird der Wert 2,6 einer Integer-Variablen zugewiesen, die anschließend den Wert 3 besitzt.

Zweites Beispiel:

```
Sub DivisionMitRunden()
    Dim x As Integer
    x = 13 / 5
    MsgBox x
End Sub
```

Listing 3.7 Sub »DivisionMitRunden()« in Mappe3.xlsm, Modul 1

Ganzzahldivision

Die Ganzzahldivision wird in zwei Schritten durchgeführt. Im ersten Schritt werden Dividend und Divisor einzeln gerundet. Im zweiten Schritt werden die beiden verbliebenen ganzen Zahlen geteilt. Anschließend werden die Ziffern nach dem Komma abgeschnitten. Tabelle 3.3 zeigt einige Beispiele.

Ausdruck	Ergebnis
19 / 4	4,75
19 \ 4	4
19 \ 4.6	3
19.5 \ 4.2	5

Tabelle 3.3 Ganzzahldivision

Modulo-Operator

Der Modulo-Operator `Mod`berechnet den Rest einer Division. Diese Berechnung wird ebenfalls in zwei Schritten durchgeführt. Im ersten Schritt werden Dividend und Divisor einzeln gerundet. Im zweiten Schritt wird der Rest der Ganzzahldivision ermittelt. Einige Beispiele sehen Sie in Tabelle 3.4.

Ausdruck	Ergebnis
19 Mod 4	3
19.5 Mod 3.2	2

Tabelle 3.4 Modulo-Operator

In Excel gibt es die Funktion `REST()`. Sie berechnet den Rest, ohne vorher zu runden. Die Berechnung des zweiten Beispiels aus Tabelle 3.4 mit `REST(19,5; 3,2)` ergibt damit 0,3 statt 2.

Potenzierung

Der Potenzierung einer Zahl dient in Excel der Operator ^ (hoch). Diesen Operator gibt es ebenso in VBA. Außerdem gibt es die Funktion `POTENZ()`. Mit beiden werden dieselben Ergebnisse wie mit dem VBA-Operator ^ ermittelt (Beispiele siehe Tabelle 3.5).

Ausdruck	Ergebnis
`2 ^ 5`	32
`3 ^ 2 ^ 3`	729
`2 ^ 5.4`	42,2242531447326
`-2 ^ 5`	–32

Tabelle 3.5 Operator ^ (hoch)

Eine Beispielprozedur, in der eine Reihe von arithmetischen Operatoren eingesetzt wird:

```
Sub ArithmetischeOperatoren()
    ThisWorkbook.Worksheets("Tabelle1").Activate

    Range("C1").Value = 5 * 3 - 6 / 3
    Range("C2").Value = 5 * (3 - 6) / 3

    Range("C4").Value = 19 / 4
    Range("C5").Value = 19 \ 4
    Range("C6").Value = 19 \ 4.6
    Range("C7").Value = 19.5 \ 4.2

    Range("D1").Value = 19 Mod 4
    Range("D2").Value = 19.5 Mod 3.2

    Range("D4").Value = 2 ^ 5
    Range("D5").Value = 3 ^ 2 ^ 3
    Range("D6").Value = 2 ^ 5.4
    Range("D7").Value = -2 ^ 5
End Sub
```

Listing 3.8 Sub »ArithmetischeOperatoren()« in Mappe3.xlsm, Modul 1

Das Ergebnis der Berechnungen sehen Sie in Abbildung 3.5.

C	D
13	3
-5	2
4,75	32
4	729
3	42,2242531
5	-32

Abbildung 3.5 Ergebnis der Berechnungen

Multiplikation und Division sind innerhalb eines Ausdrucks gleichrangig und werden von links nach rechts in der Reihenfolge ihres Auftretens ausgewertet. Dasselbe gilt für Addition und Subtraktion innerhalb eines Ausdrucks.

Mit Klammern wird diese Rangfolge außer Kraft gesetzt. Operationen in Klammern haben grundsätzlich Vorrang.

3.3.2 Vergleichsoperatoren

Vergleichsoperatoren werden in Bedingungen benötigt. Diese wiederum nutzen Sie für Verzweigungen und Schleifen, die der Ablaufsteuerung von Programmen dienen. Verzweigungen arbeiten nach dem gleichen Prinzip wie die Funktion `WENN()` in Excel.

Das Ergebnis einer Bedingung ist einer der beiden Werte, die für eine Variable des Typs `Boolean` verwendet werden dürfen: `True` (wahr) oder `False` (falsch). Tabelle 3.6 führt die Vergleichsoperatoren auf.

Operator	Beschreibung
`<`	kleiner als
`<=`	kleiner als oder gleich
`>`	größer als
`>=`	größer als oder gleich
`=`	gleich
`<>`	ungleich

Tabelle 3.6 Vergleichsoperatoren

Einige Beispiele sehen Sie in Tabelle 3.7.

Ausdruck	Ergebnis
`5 > 3`	wahr
`3 = 3.2`	falsch
`5 + 3 * 2 >= 12`	falsch

Tabelle 3.7 Ausdrücke mit Vergleichsoperatoren

Darüber hinaus gibt es den Operator `Like`, der dem Mustervergleich dient. Dabei setzen Sie u. a. die Platzhalter * (Stern) für eines oder mehrere Zeichen und ? (Fragezeichen) für genau ein Zeichen ein. Tabelle 3.8 zeigt einige Beispiele.

Ausdruck	Ergebnis
`"abxba" Like "a*a"`	wahr
`"abxba" Like "a?a"`	falsch
`"aba" Like "a?a"`	wahr
`"asdlfigc" Like "a?d?f*c"`	wahr

Tabelle 3.8 Operator »Like«

Mithilfe von `Like` können Sie noch wesentlich komplexere Mustervergleiche durchführen als in den hier genannten Beispielen.

Eine Beispielprozedur:

```
Sub VergleichsOperatoren()
    ThisWorkbook.Worksheets("Tabelle1").Activate

    Range("E1").Value = (5 > 3)
    Range("E2").Value = (3 = 3.2)
    Range("E3").Value = (5 + 3 * 2 >= 12)

    Range("F1").Value = ("abxba" Like "a*a")
    Range("F2").Value = ("abxba" Like "a?a")
    Range("F3").Value = ("aba" Like "a?a")
    Range("F4").Value = ("asdlfigc" Like "a?d?f*c")
End Sub
```

Listing 3.9 Sub »VergleichsOperatoren()« in Mappe3.xlsm, Modul 1

Hinweis

Die Klammern um die Ausdrücke mit den Vergleichsoperatoren sind für die richtige Auswertung nicht notwendig. Sie dienen der besseren Lesbarkeit.

Das Ergebnis der Vergleiche sehen Sie in Abbildung 3.6.

E	F
WAHR	WAHR
FALSCH	FALSCH
FALSCH	WAHR
	WAHR

Abbildung 3.6 Ergebnis der Vergleiche

Übung »Vergleichsoperatoren«

Schreiben Sie die Prozedur `UebungVergleich()`. Ermitteln Sie darin das Ergebnis der beiden folgenden Ausdrücke, speichern Sie es in einer Variablen eines geeigneten Datentyps, und zeigen Sie es an:

```
12 - 3 >= 4 * 2.5
"Maier" Like "M??er"
```

3.3.3 Logische Operatoren

Logische Operatoren dienen dazu, mehrere Bedingungen zusammenzufassen. Sie werden in Tabelle 3.9 aufgelistet.

Operator	Beschreibung	Das Ergebnis ist wahr, wenn ...
`Not`	Nicht	... der Ausdruck falsch ist.
`And`	Und	... beide Ausdrücke wahr sind.
`Or`	inklusives Oder	... mindestens ein Ausdruck wahr ist.
`Xor`	exklusives Oder	... genau ein Ausdruck wahr ist.

Tabelle 3.9 Logische Operatoren

In Tabelle 3.10 sehen Sie einige Beispiele. Es gilt `a = 1`, `b = 3` und `c = 5`.

Ausdruck	Ergebnis
`Not (a < b)`	falsch
`(b > a) And (c > b)`	wahr
`(b < a) Or (c < b)`	falsch
`(b < a) Xor (c > b)`	wahr

Tabelle 3.10 Ausdrücke mit logischen Operatoren

Die Funktionsweise der logischen Operatoren entnehmen Sie Tabelle 3.11.

Ausdruck 1	Ausdruck 2	Not (Ausdruck 1)	And	Or	Xor
wahr	wahr	falsch	wahr	wahr	falsch
wahr	falsch	falsch	falsch	wahr	wahr
falsch	wahr	wahr	falsch	wahr	wahr
falsch	falsch	wahr	falsch	falsch	falsch

Tabelle 3.11 Funktionsweise logischer Operatoren

Eine Beispielprozedur:

```
Sub LogischeOperatoren()
    Dim a As Integer, b As Integer, c As Integer
    a = 1
    b = 3
    c = 5

    ThisWorkbook.Worksheets("Tabelle1").Activate
    Range("G1").Value = Not (a < b)
    Range("G2").Value = (b > a) And (c > b)
    Range("G3").Value = (b < a) Or (c < b)
    Range("G4").Value = (b < a) Xor (c > b)
End Sub
```

Listing 3.10 Sub »LogischeOperatoren()« in Mappe3.xlsm, Modul 1

Das Ergebnis der Vergleiche sehen Sie in Abbildung 3.7.

G
FALSCH
WAHR
FALSCH
WAHR

Abbildung 3.7 Ergebnis der Vergleiche mit logischen Operatoren

> **Übung »Logische Operatoren«**
>
> Schreiben Sie die Prozedur `UebungLogisch()`. Ermitteln Sie darin das Ergebnis der beiden folgenden Ausdrücke, speichern Sie es in einer Variablen eines geeigneten Datentyps, und zeigen Sie es an:
>
> `4 > 3 And -4 > -3`
>
> `4 > 3 Or -4 > -3`

3.3.4 Verkettungsoperator

Der Operator & dient der Verkettung von Zeichenketten. Er wird häufig benötigt, um Ergebnisse anschaulich zusammen mit Text darzustellen. Ist einer der Ausdrücke keine Zeichenfolge, sondern eine Zahl oder Datumsangabe, wird er in eine Zeichenfolge verwandelt. Das Gesamtergebnis ist dann wiederum eine Zeichenfolge. Ein Beispiel:

```
Sub VerkettungsOperator()
    Dim a As String
    Dim s As Single
    Dim d As Date
    d = "15.08.2022"
    s = 4.6
    a = "Temperatur: " & s & " Grad am " & d
    MsgBox a
End Sub
```

Listing 3.11 Sub »VerkettungsOperator()« in Mappe3.xlsm, Modul 1

Das Ergebnis der Verkettung sehen Sie in Abbildung 3.8.

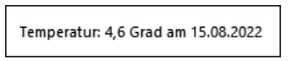


Abbildung 3.8 Verkettung unterschiedlicher Daten

3.3.5 Rangfolge der Operatoren

Enthält ein Ausdruck mehrere Operatoren, werden die einzelnen Teilausdrücke gemäß der *Rangfolge* bzw. *Priorität der Operatoren* ausgewertet (siehe Tabelle 3.12).

Rang	Operator	Beschreibung
1	^	Potenzierung (Exponentialoperator)
2	-	negatives Vorzeichen
3	* und /	Multiplikation und Division
4	\	Ganzzahldivision
5	Mod	Modulo
6	+ und -	Addition und Subtraktion
7	&	Verkettung
8	=, <>, <, >, <=, >= und Like	Vergleichsoperatoren (das Zeichen = steht für den Vergleich, nicht für die Zuweisung)
9	Not	logisches Nicht
10	And	logisches Und
11	Or	logisches Oder
12	Xor	logisches Oder (exklusiv)

Tabelle 3.12 Rangfolge der Operatoren

Ausdrücke mit Operatoren der gleichen Priorität werden von links nach rechts ausgewertet. Mit Klammern wird die Rangfolge außer Kraft gesetzt. Operationen in Klammern haben grundsätzlich Vorrang.

> **Übung »Rangfolge der Operatoren«**
>
> Sind die Bedingungen in Tabelle 3.13 wahr oder falsch? Lösen Sie die Aufgabe möglichst auf dem Papier, ohne VBA. Zum Vergleich finden Sie eine Lösung in der Prozedur `Uebung-Rangfolge()` in *Mappe3.xlsm*, Modul 2.

Nr.	Werte	Bedingung
1	a=5 b=10	a>0 And b<>10
2	a=5 b=10	a>0 Or b<>10
3	z=10 w=100	z<>0 Or z>w Or w-z=90
4	z=10 w=100	z=11 And z>w Or w-z=90
5	x=1.0 y=5.7	x>=.9 And y<=5.8
6	x=1.0 y=5.7	x>=.9 And Not(y<=5.8)
7	n1=1 n2=17	n1>0 And n2>0 Or n1>n2 And n2<>17
8	n1=1 n2=17	n1>0 And (n2>0 Or n1>n2) And n2<>17

Tabelle 3.13 Angaben zu Übung »Rangfolge der Operatoren«

3.4 Verzweigungen

Der Programmcode wurde bisher sequenziell bearbeitet, d. h. eine Anweisung nach der anderen. Kontrollstrukturen ermöglichen eine Steuerung der Bearbeitung. Sie unterteilen sich in Verzweigungen und Schleifen.

Verzweigungen gestatten dem Programm, unterschiedliche Anweisungsblöcke auszuführen. Es gibt die beiden Verzweigungsstrukturen `If ... Then ... Else` und `Select Case ...` Dabei wird die Programmausführung aufgrund von Bedingungen an einen bestimmten Anweisungsblock übergeben. Bedingungen werden mithilfe von Vergleichsoperatoren erstellt.

Verzweigungen arbeiten nach dem gleichen Prinzip wie die Funktion `WENN()` in Excel. Gibt es mehr als zwei Möglichkeiten, kann diese Funktion verschachtelt werden. Dies trifft auf die Verzweigungen in VBA ebenfalls zu.

3.4.1 Einzeiliges »If ... Then ... Else«

Das einzeilige `If ... Then ... Else` hat folgenden Aufbau:

```
If Bedingung Then Anweisungen1 [ Else Anweisungen2 ]
```

Bedingungen werden ausgewertet und ergeben entweder wahr oder falsch. Ist das Ergebnis wahr, wird der `Then`-Teil mit den `Anweisungen1` ausgeführt. Ist das Ergebnis falsch und gibt es einen `Else`-Teil, wird der `Else`-Teil mit den `Anweisungen2` ausgeführt.

In der folgenden Prozedur wird das einzeilige `If` in zwei verschiedenen Beispielen genutzt:

```
Sub EinzeilenIf()
    ThisWorkbook.Worksheets("Tabelle1").Activate
    If Range("A1").Value > 100 Then MsgBox "Groß"
    If Range("A1").Value < 10 Then MsgBox "Klein" Else MsgBox "Nicht klein"
End Sub
```

Listing 3.12 Sub »EinzeilenIf()« in Mappe3.xlsm, Modul 1

Für das erste Beispiel gilt: Ist der Wert in der Zelle größer als 100, erfolgt eine Ausgabe, anderenfalls passiert nichts. Beim zweiten Beispiel wird in jedem Fall etwas ausgegeben.

3.4.2 If-Then-Else-Block

Für die Ausführung von einzelnen Anweisungen ist das einzeilige `If` geeignet. Sind mehrere Anweisungen auszuführen, wird der Programmcode schnell unübersichtlich. Dann ist ein `If-Then-Else`-Block besser geeignet. Er hat folgenden Aufbau:

```
If Bedingung1 Then
   Anweisungen1
[ ElseIf Bedingung2
   Anweisungen2 ] ...
[ Else
   AnweisungenX ]
End If
```

Zunächst wird die erste Bedingung geprüft. Trifft sie nicht zu, wird die zweite Bedingung geprüft usw. Das Programm verzweigt zu den Anweisungen hinter der ersten zutreffenden Bedingung. Trifft keine der Bedingungen zu, werden die Anweisungen hinter dem `Else` ausgeführt, sofern es diesen `Else`-Zweig gibt. Anderenfalls wird keine Anweisung durchgeführt. Ein `If-Then-Else`-Block endet immer mit einem `End If`.

In der folgenden Prozedur werden mithilfe von zwei Bedingungen drei verschiedene Fälle unterschieden:


```
Sub BlockIf()
    ThisWorkbook.Worksheets("Tabelle1").Activate

    If Range("C1").Value > 100 Then
        Range("C1").Font.Size = 14
        Range("C1").Font.Italic = True
        Range("C1").Font.Underline = True
    ElseIf Range("C1").Value < 10 Then
        Range("C1").Font.Size = 11
        Range("C1").Font.Italic = False
        Range("C1").Font.Underline = False
    Else
        Range("C1").Font.Size = 17
        Range("C1").Font.Italic = False
        Range("C1").Font.Underline = True
    End If
End Sub
```

Listing 3.13 Sub »BlockIf()« in Mappe3.xlsm, Modul 1

Das Ergebnis der Verzweigung sehen Sie in Abbildung 3.9.

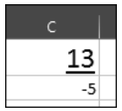
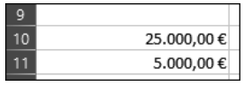


Abbildung 3.9 Der dritte Fall trifft zu.

Je nach Wert der Zelle werden die Schrifteigenschaften »Größe«, »kursiv« und »unterstrichen« eingestellt. Hier ist der Wert weder größer als 100 noch kleiner als 10. Daher trifft der dritte Fall zu.

> **Übung »Verzweigung mit If«**
>
> Schreiben Sie in der Prozedur `UebungIf()` ein stark vereinfachtes Programm zur Berechnung der Steuer. Zu einem Gehaltsbetrag, der in Zelle `A10` steht, wird der Steuerbetrag berechnet und in der Zelle darunter ausgegeben. Formatieren Sie beide Zellen wie in Abbildung 3.10 gezeigt.



9	
10	25.000,00 €
11	5.000,00 €

Abbildung 3.10 Beispielwert einschließlich Ergebnis

In Tabelle 3.14 werden die Steuersätze für die jeweiligen Gehälter genannt. Verwenden Sie für die Verzweigung einen `If-Then-Else`-Block.

Gehalt	Steuersatz
bis einschließlich 12.000 €	12 %
größer als 12.000 bis einschließlich 20.000 €	15 %
größer als 20.000 bis einschließlich 30.000 €	20 %
größer als 30.000 €	25 %

Tabelle 3.14 Angaben zu Übung »Verzweigung mit If«

3.4.3 Select Case

Die Anweisung `Select Case ...` dient als Alternative zum `If-Then-Else`-Block. Sie ermöglicht die übersichtlichere Gestaltung einer Mehrfachauswahl und ist wie folgt aufgebaut:

```
Select Case Testausdruck
   [ Case Ausdrucksliste1
      Anweisungen1 ]
   [ Case Ausdrucksliste2
      Anweisungen2 ] ...
   [ Case Else
      AnweisungenX ]
End Select
```

Die Anweisung `Select Case ...` verwendet einen Testausdruck, der am Beginn der Struktur ausgewertet wird. Sein Wert wird anschließend der Reihe nach mit den Werten der Ausdrucksklisten verglichen.

Eine Ausdrucksliste kann aus mehreren Ausdrücken oder einer Bereichsangabe mit dem Schlüsselwort `To` bestehen. Ein Ausdruck kann aus einem Wert oder einer Bedingung mit dem Schlüsselwort `Is` bestehen.

Bei der ersten Übereinstimmung wird der zugehörige Anweisungsblock ausgeführt und dann mit der nächsten Anweisung hinter dem `End Select` fortgefahren.

Wird keine Übereinstimmung gefunden, wird der optionale Anweisungsblock hinter dem `Case Else` ausgeführt.

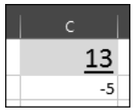
In der folgenden Prozedur werden verschiedene Fälle geprüft. Trifft keiner dieser Fälle zu, wird der `Case-Else`-Zweig ausgeführt:

```
Sub SelectCase()
    ThisWorkbook.Worksheets("Tabelle1").Activate

    Select Case Range("C1").Value
       Case 20, 30, 40
          Range("C1").Interior.Color = vbRed
       Case Is < 10, Is > 100
          Range("C1").Interior.Color = vbGreen
       Case 12 To 17
          Range("C1").Interior.Color = vbCyan
       Case Else
          Range("C1").Interior.Color = vbYellow
    End Select
End Sub
```

Listing 3.14 Sub »SelectCase()« in Mappe3.xlsm, Modul 1

Das Ergebnis der Mehrfachauswahl sehen Sie in Abbildung 3.11.



C
13
-5

Abbildung 3.11 Der dritte Fall trifft zu.

Der Ausdruck `Case 20, 30, 40` bedeutet: »Ist der Wert 20, 30 oder 40 ...« Als Nächstes folgt die Ausdrucksliste `Case Is < 10, Is > 100`. Sie bedeutet: »Ist der Wert kleiner als 10 oder größer als 100 ...«

Der Ausdruck `Case 12 To 17` bedeutet: »Ist der Wert größer als oder gleich 12 und kleiner als oder gleich 17 ...«, was hier zutrifft. Der Ausdruck `Case Else` steht für die restlichen Fälle.

> **Übung »Verzweigung mit Select«**
>
> Schreiben Sie in der Prozedur `UebungSelect()` das Programm aus der vorherigen Übung »Verzweigung mit If«. Benutzen Sie hierfür die Verzweigung `Select Case`.

3.5 Schleifen

Schleifen ermöglichen den mehrfachen Durchlauf von Anweisungen. Darin liegt eine besondere Stärke der Programmierung allgemein: die schnelle wiederholte Bearbeitung ähnlicher Vorgänge.

Es gibt die folgenden Schleifenstrukturen:

- `Do ... Loop`
- `For ... Next`
- `For Each ... In ... Next`

Mithilfe der ersten beiden Strukturen steuern Sie die Wiederholungen eines Anweisungsblocks, also die Durchläufe der Schleife. Dabei wird der Wahrheitswert eines logischen Ausdrucks (der Schleifenbedingung) oder der Wert eines numerischen Ausdrucks (der Schleifenvariablen) benötigt.

Eine Schleife mit `For Each ... In ... Next` wird im Zusammenhang mit Auflistungen oder Feldern eingesetzt. Mehr dazu in Abschnitt 6.5, »For-Each-Schleife«.

3.5.1 Schleife mit »For ... Next«

Ist die Anzahl der Schleifendurchläufe bekannt oder vor Beginn der Schleife berechenbar, wird eine Schleife mit `For ... Next` verwendet.

Ihr Aufbau sieht wie folgt aus:

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

Die Schleifenvariable `Zahl` wird zunächst auf den Wert von `Anfang` gesetzt. Nach jedem Durchlauf wird sie um den Wert von `Schritt` verändert, also vergrößert oder verkleinert. Wird `Step = Schritt` nicht angegeben, wird die Variable um 1 vergrößert. Der neue Wert von `Zahl` wird mit dem Wert von `Ende` verglichen:

- Ist die Schrittweite positiv und der Wert von `Zahl` größer als der Wert von `Ende`, wird die Schleife beendet, ansonsten nochmals durchlaufen.
- Ist die Schrittweite negativ und der Wert von `Zahl` kleiner als der Wert von `Ende`, wird die Schleife auch beendet, ansonsten ebenso nochmals durchlaufen.

Die Anweisung `Exit For` wird eingesetzt, um die Schleife aufgrund einer besonderen Bedingung unmittelbar zu verlassen.

In den folgenden Prozeduren werden drei unterschiedliche Schleifen durchlaufen. Die Ausgaben erfolgen im Tabellenblatt `Tabelle2`. Stellen Sie vor der Ausführung sicher, dass es existiert. Das erste Beispiel:

```
Sub ForNext1()
    Dim i As Integer
    ThisWorkbook.Worksheets("Tabelle2").Activate
    Range("B1:B10").Clear

    For i = 3 To 7
        Cells(i, 2).Value = Cells(i, 1).Value * 2
    Next i
End Sub
```

Listing 3.15 Sub »ForNext1()« in Mappe3.xlsm, Modul 1

Abbildung 3.12 zeigt das Ergebnis.

	A	B
1	1	
2	2	
3	3	6
4	4	8
5	5	10
6	6	12
7	7	14
8	8	
9	9	
10	10	

Abbildung 3.12 Schleife mit »i« von 3 bis 7, Schrittweite 1

Die Zahlen 1 bis 10 in Spalte `A` sind bereits vorhanden. Der Zielbereich wird vorher mit der Methode `Clear()` gelöscht, da auch die nachfolgenden Ergebnisse hier notiert werden. Als Schleifenvariable dient `i`. Die Schleife wird erstmalig mit `i = 3` und letztmalig mit `i = 7` durchlaufen. Es ist keine Schrittweite angegeben, daher beträgt sie 1.

Ein weiteres Beispiel:

```
Sub ForNext2()
    Dim i As Integer
    ThisWorkbook.Worksheets("Tabelle2").Activate
```

```
    Range("B1:B10").Clear

    For i = 3 To 7 Step 2
        Cells(i, 2).Value = Cells(i, 1).Value * 2
    Next i
End Sub
```

Listing 3.16 Sub »ForNext2()« in Mappe3.xlsm, Modul 1

Das Ergebnis sehen Sie in Abbildung 3.13.

	A	B
1	1	
2	2	
3	3	6
4	4	
5	5	10
6	6	
7	7	14
8	8	
9	9	
10	10	

Abbildung 3.13 Schleife mit »i« von 3 bis 7, Schrittweite 2

Aufgrund der Schrittweite von 2 wird nur jede zweite Zelle bearbeitet.

Ein letztes Beispiel:

```
Sub ForNext3()
    Dim i As Integer
    Dim Ergebnis As Integer
    ThisWorkbook.Worksheets("Tabelle2").Activate
    Range("B1:B10").Clear

    For i = 9 To 2 Step -2
        Ergebnis = Cells(i, 1).Value * 2
        If Ergebnis < 10 Then Exit For
        Cells(i, 2).Value = Ergebnis
    Next i
End Sub
```

Listing 3.17 Sub »ForNext3()« in Mappe3.xlsm, Modul 1

Das Ergebnis ist in Abbildung 3.14 dargestellt.

Die Schleife wird, beginnend mit dem größten Wert, mit negativer Schrittweite durchlaufen. Das Rechenergebnis wird zur Verdeutlichung in der Variablen `Ergebnis` zwischengespeichert. Ist die besondere Bedingung `Ergebnis < 10` erfüllt, wird die Schleife mit `Exit For` vorzeitig verlassen.

	A	B
1	1	
2	2	
3	3	
4	4	
5	5	10
6	6	
7	7	14
8	8	
9	9	18
10	10	

Abbildung 3.14 Schleife mit »i« von 9 bis 2, Schrittweite –2, mit »Exit For«

Die Schleifen lassen sich auch mithilfe des `Range`-Objekts bilden. Nachfolgend eine Alternative für die Prozedur `ForNext1()`:

```
Sub ForNext4()
    Dim i As Integer
    ThisWorkbook.Worksheets("Tabelle2").Activate
    Range("B1:B10").Clear

    For i = 3 To 7
        Range("B" & i).Value = Range("A" & i).Value * 2
    Next i
End Sub
```

Listing 3.18 Sub »ForNext4()« in Mappe3.xlsm, Modul 1

Die Zellbezeichnungen von `B3` bis `B7` bzw. von `A3` bis `A7` werden mithilfe einer Verkettung gebildet. Mithilfe der nachfolgenden Prozedur wird eine Schleife über mehrere Spalten gebildet:

```
Sub ForNext5()
    Dim i As Integer
    ThisWorkbook.Worksheets("Tabelle2").Activate
    Range("A20:E20").Clear
```

```
    For i = 1 To 5
        Range(Chr(i + 64) & "20").Value = i
    Next i
End Sub
```

Listing 3.19 Sub »ForNext5()« in Mappe3.xlsm, Modul 1

Zur Erzeugung der Buchstaben für die Bezeichnung der Spalte wird die VBA-Funktion `Chr()` genutzt. Sie liefert zu einer Codenummer aus dem ASCII-Code das zugehörige Zeichen. Die Großbuchstaben von A bis Z haben die Codenummern 65 bis 90.

Übung »Schleife mit For«

Programmieren Sie in der Prozedur `UebungFor()` eine Schleife, mit der die in Abbildung 3.15 gezeigten Zahlen in den Zellen `D1` bis `D7` ausgegeben werden.

D
35,0
32,5
30,0
27,5
25,0
22,5
20,0

Abbildung 3.15 Ergebnis zu Übung »Schleife mit For«

3.5.2 Schleife mit »Do ... Loop«

Ist die Anzahl der Schleifendurchläufe nicht bekannt bzw. vor Beginn der Schleife nicht berechenbar, wird die Schleife mit `Do ... Loop` verwendet. Es gibt fünf verschiedene Typen, und zwar mit:

- `Do While ... Loop`: Die Bedingung zum Durchlauf der Schleife wird am Anfang geprüft.
- `Do ... Loop While`: Die Bedingung zum Durchlauf der Schleife wird am Ende geprüft.
- `Do Until ... Loop`: Die Bedingung zum Abbruch der Schleife wird am Anfang geprüft.
- `Do ... Loop Until`: Die Bedingung zum Abbruch der Schleife wird am Ende geprüft.
- `Do ... Loop`: Es gibt keine Bedingung zum Durchlauf oder Abbruch der Schleife. Zur Beendigung der Schleife sind eine Verzweigung in der Schleife und ein `Exit Do` notwendig. Ansonsten würde die Schleife endlos weiterlaufen.

Die Schleifen mit `Do ... Loop While` und `Do ... Loop Until` nennt man auch fußgesteuert: Die Bedingung wird erst am Fuß der Schleife geprüft. Eine solche Schleife wird mindestens einmal durchlaufen.

Im Gegensatz dazu gibt es die kopfgesteuerten Schleifen mit `Do While ... Loop` und `Do Until ... Loop`: Die Bedingung wird bereits im Kopf der Schleife geprüft. Dabei kann es vorkommen, dass diese Schleife niemals durchlaufen wird.

Der allgemeine Aufbau sieht wie folgt aus:

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

oder

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

In den folgenden Prozeduren werden drei der fünf Möglichkeiten genutzt. Es werden jeweils so lange Zahlen addiert, bis die Summe der Zahlen 5 erreicht. Da die Zahlen durch einen Zufallszahlengenerator erzeugt werden, ist die Anzahl der Schleifendurchläufe nicht vorhersagbar.

Hinweis

Der Zufallszahlengenerator wird mithilfe der Funktion `Rnd()` realisiert. Diese liefert zufällige Zahlen mit einem Wert, der größer als oder gleich 0 und kleiner als 1 ist. Beim Aufruf der Funktion `Rnd()` fallen die Klammern weg, da sie keine Parameter besitzt.

Der Zufallszahlengenerator sollte vor der ersten Benutzung mithilfe der Anweisung `Randomize` initialisiert werden. Ansonsten werden immer wieder die gleichen Zahlenfolgen geliefert, die dann eben nicht mehr zufällig sind.

Das erste Beispiel:

```
Sub DoLoop1()
    Dim i As Integer
    Dim Summe As Single

    ThisWorkbook.Worksheets("Tabelle2").Activate
    Range("C1:C20").Clear
    Randomize

    i = 1
    Summe = 0
    Do
        Summe = Summe + Rnd
        Cells(i, 3).NumberFormatLocal = "0,000"
        Cells(i, 3).Value = Summe
        i = i + 1
    Loop While Summe < 5

    Cells(i, 3).Value = "Fertig"
End Sub
```

Listing 3.20 Sub »DoLoop1()« in Mappe3.xlsm, Modul 1

Der Zielbereich wird zu Beginn mit der Methode `Clear()` gelöscht, da die Ergebnisse aller drei Beispiele hier erscheinen. Der Zufallszahlengenerator wird initialisiert.

Vor der Schleife wird die Variable `i` für die Zeilennummer der Ausgabezelle auf 1 gesetzt. Die Variable `Summe` wird mit dem Wert 0 initialisiert. Dies ist nicht notwendig, gehört aber zum guten Programmierstil. Damit verdeutlichen Sie den Startwert der Summe.

In der Schleife wird der Wert der Variablen `Summe` um eine Zufallszahl zwischen 0 und 1 erhöht. Die Summe wird ausgegeben. Die Variable `i` für die Zeilennummer der Ausgabezelle wird um 1 erhöht, damit der nächste Wert von `Summe` in der nächsten Zeile ausgegeben wird.

Am Ende der Schleife wird mithilfe von `While` geprüft, ob die Summe kleiner als 5 ist. Trifft dies zu, läuft die Schleife weiter, andernfalls wird sie beendet. Am Ende der Prozedur wird die Ausgabe »Fertig« erzeugt, siehe Abbildung 3.16.

C
0,035
0,263
0,587
1,536
2,057
2,956
3,125
3,211
3,558
4,518
5,135
Fertig

Abbildung 3.16 Addition zufälliger Zahlen bis zur Grenzsumme 5

Das zweite Beispiel:

```
Sub DoLoop2()
    .....
    Do
        Summe = Summe + Rnd
        Cells(i, 3).NumberFormatLocal = "0,000"
        Cells(i, 3).Value = Summe
        i = i + 1
    Loop Until Summe >= 5

    Cells(i, 3).Value = "Fertig"
End Sub
```

Listing 3.21 Sub »DoLoop2()« in Mappe3.xlsm, Modul 1

Der Anfang dieser Prozedur entspricht dem Anfang der vorherigen Prozedur. Am Ende der Schleife wird mithilfe von `Until` geprüft, ob die Summe größer als oder gleich 5 ist. Trifft dies zu, wird die Schleife beendet, andernfalls läuft sie weiter.

Das dritte und letzte Beispiel:

```
Sub DoLoop3()
    .....
    Do
        Summe = Summe + Int(Rnd * 6 + 1)
        Cells(i, 3).Value = Summe
```

```
        i = i + 1
        If Summe >= 30 Then Exit Do
    Loop

    Cells(i, 3).Value = "Fertig"
End Sub
```

Listing 3.22 Sub »DoLoop3()« in Mappe3.xlsm, Modul 1

Der Anfang dieser Prozedur ist wieder gleich geblieben. In der Schleife wird mithilfe einer Verzweigung geprüft, ob die Summe größer als oder gleich 30 ist. Trifft dies zu, wird die Schleife mit `Exit Do` abgebrochen, andernfalls läuft sie weiter.

In diesem dritten Beispiel wird mit zufälligen ganzen Zahlen gearbeitet. Die Funktion `Rnd()` liefert Zahlen mit einem Wert, der größer als oder gleich 0 und kleiner als 1 ist. Nach der Multiplikation mit 6 sind diese Werte größer als oder gleich 0 und kleiner als 6. Nach der Addition von 1 sind diese Werte größer als oder gleich 1 und kleiner als 7. Die Funktion `Int()` schneidet die Nachkommastellen ab. Es ergeben sich zufällige ganze Zahlen von 1 bis 6, also die Zahlen eines Würfels.

Übung »Schleife mit Do«

Schreiben Sie die Prozedur `UebungDo()`, mit deren Hilfe eine Zahl, die in der obersten Zelle einer Spalte steht, wiederholt halbiert wird. Alle Zahlen werden, wie in Abbildung 3.17 dargestellt, nacheinander unter der ursprünglichen Zahl ausgegeben. Ist das Ergebnis der Halbierung kleiner als 0,001, wird die Prozedur beendet.

E
2,000
1,000
0,500
0,250
0,125
0,063
0,031
0,016
0,008

Abbildung 3.17 Ergebnis zu Übung »Schleife mit Do«

Übung »Steuertabelle«

In der Übung »Verzweigung mit If« (siehe Abschnitt 3.4.2, »If-Then-Else-Block«) haben Sie ein stark vereinfachtes Programm zur Berechnung der Steuer geschrieben. Erweitern Sie die Lösung, indem Sie in der Prozedur `UebungTabelle()` zu einer Reihe von Gehältern u. a. den Steuerbetrag berechnen und ausgeben.

In Tabelle 3.15 sind die Steuersätze angegeben.

Gehalt	Steuersatz
bis einschließlich 12.000 €	12 %
größer als 12.000 bis einschließlich 20.000 €	15 %
größer als 20.000 bis einschließlich 30.000 €	20 %
größer als 30.000 €	25 %

Tabelle 3.15 Angaben zu Übung »Steuertabelle«

Es werden für jedes Gehalt von 9.000 € bis 33.000 € in Schritten von 3.000 € folgende vier Werte ausgegeben: das Gehalt, der Steuersatz, der Steuerbetrag und das Gehalt abzüglich des Steuerbetrags.

Jedes Gehalt wird mit den zugehörigen Werten in einer Zeile ausgegeben, siehe Abbildung 3.18.

	A	B	C	D	E
1	9.000,00 €	12,0 %	1.080,00 €	7.920,00 €	
2	12.000,00 €	12,0 %	1.440,00 €	10.560,00 €	
3	15.000,00 €	15,0 %	2.250,00 €	12.750,00 €	
4	18.000,00 €	15,0 %	2.700,00 €	15.300,00 €	
5	21.000,00 €	20,0 %	4.200,00 €	16.800,00 €	
6	24.000,00 €	20,0 %	4.800,00 €	19.200,00 €	
7	27.000,00 €	20,0 %	5.400,00 €	21.600,00 €	
8	30.000,00 €	20,0 %	6.000,00 €	24.000,00 €	
9	33.000,00 €	25,0 %	8.250,00 €	24.750,00 €	

Abbildung 3.18 Ergebnis zu Übung »Steuertabelle«

Übung »Kopfrechnen mit Verzweigung«

Schreiben Sie ein Programm, mit dessen Hilfe die Benutzerin das Kopfrechnen trainiert. Nach dem Betätigen der Schaltfläche AUFGABE STELLEN (siehe Abbildung 3.19) wird die Prozedur `UebungKopfVerzweigung_Aufgabe()` aufgerufen. Mit ihrer Hilfe erscheint in den Zellen `A1` bis `D1` eine Aufgabenstellung. Außerdem wird das richtige Ergebnis berechnet und in Zelle `Z1` geschrieben, außerhalb des Sichtbereichs der Benutzerin. Das dient zum späteren Vergleich mit seiner Eingabe.

Die Benutzerin addiert die beiden angezeigten Zahlen im Kopf und gibt das Ergebnis in Zelle `E1` ein. Nach dem Betätigen der Schaltfläche EINGABE PRÜFEN wird die Prozedur `UebungKopfVerzweigung_Pruefen()` aufgerufen. Mit ihrer Hilfe wird die Benutzereingabe mit dem richtigen Ergebnis verglichen. Es erscheint eine Nachrichtenbox, entweder mit dem Text »Richtig« oder mit dem Text »Falsch« und dem richtigen Ergebnis, siehe Abbildung 3.20.

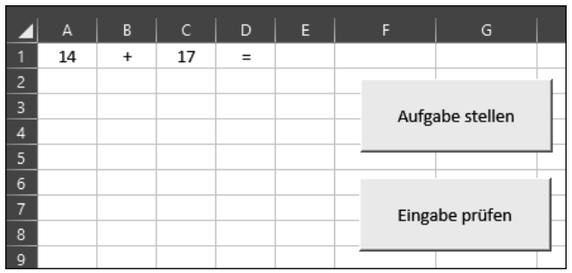


Abbildung 3.19 Aufgabenstellung

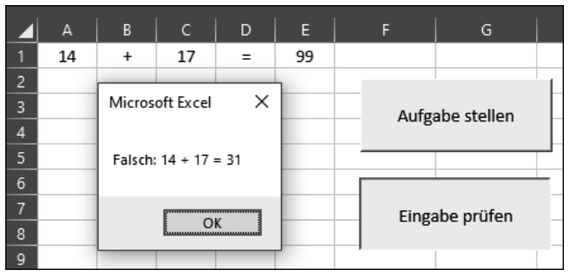


Abbildung 3.20 Reaktion auf eingetragenes Ergebnis

Die beiden Zahlen sind zufällige ganze Zahlen aus dem Bereich von 1 bis 20. In Abschnitt 1.5, »Makro per Schaltfläche ausführen«, habe ich beschrieben, wie Sie eine Prozedur mithilfe einer Schaltfläche starten.

Übung »Kopfrechnen mit Schleife«

Erweitern Sie das Programm aus der Übung »Kopfrechnen mit Verzweigung«. Nach dem Betätigen der Schaltfläche AUFGABEN STELLEN wird die Prozedur `UebungKopfSchleife_Aufgabe()` aufgerufen. Mit ihrer Hilfe erscheinen in den Zellen `A1` bis `D5` fünf verschiedene Aufgabenstellungen, siehe Abbildung 3.21. Außerdem werden die richtigen Ergebnisse berechnet und in die Zellen `Z1` bis `Z5` geschrieben.

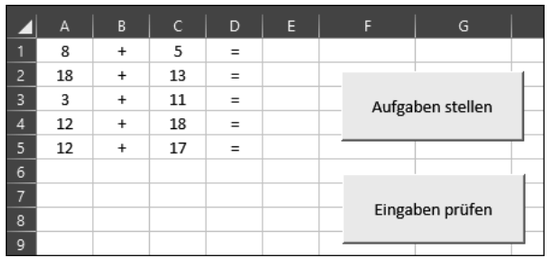


Abbildung 3.21 Fünf Aufgabenstellungen

Der Benutzer führt die fünf Additionen im Kopf durch und trägt die Ergebnisse in den Zellen `E1` bis `E5` ein. Nach dem Betätigen der Schaltfläche EINGABEN PRÜFEN wird die Prozedur `UebungKopfSchleife_Pruefen()` aufgerufen. Mit ihrer Hilfe erscheint eine Nachrichtenbox mit dem Text »Richtig:« und der Anzahl der richtig gelösten Aufgaben, siehe Abbildung 3.22.

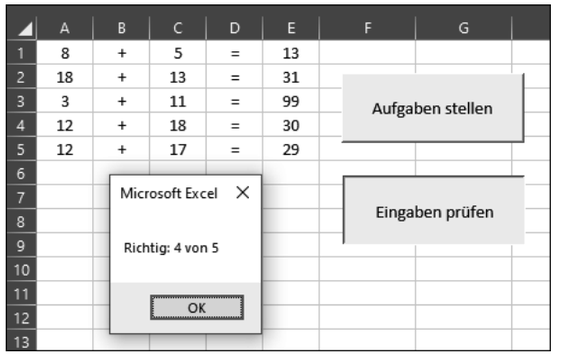


Abbildung 3.22 Reaktion auf eingetragene Ergebnisse