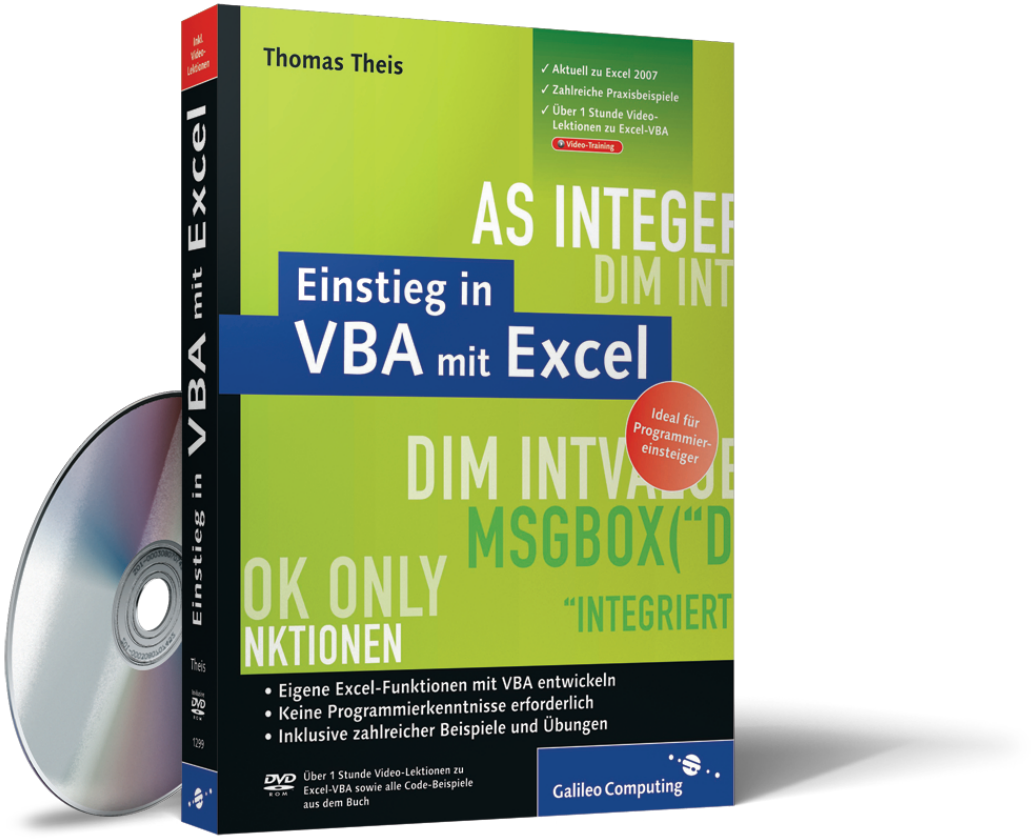
Thomas Theis

Einstieg in VBA mit Excel



Galileo Press

In diesem Kapitel lernen Sie den Umgang mit den Objekten »Arbeitsmappe«, »Tabellenblatt« und »Zellbereich« mit ihren jeweiligen Eigenschaften, Methoden und Ereignissen kennen.

2 Grundlagen von Objekten und Ereignissen

VBA ist eine objektorientierte Sprache, das heißt, es wird mit Objekten gearbeitet. Objekte verfügen über Eigenschaften, Methoden und Ereignisse. **Objekte**

- Eigenschaften (Attribute) bestimmen das Aussehen eines Objekts. Ein Tabellenblatt verfügt z. B. über die Eigenschaft `Name`. Der Wert dieser Eigenschaft ist die Bezeichnung des Tabellenblatts (z. B. *Tabelle1*). **Eigenschaft**
- Methoden bestimmen die Fähigkeiten eines Objekts. Ein Tabellenblatt verfügt z. B. über die Methode `Copy()`, das heißt, es kann kopiert werden. **Methode**
- Ereignisse bestimmen, was mit dem Objekt passiert. Bei einem Tabellenblatt kann z. B. das Ereignis *Aktivierung* stattfinden. Ereignisse können mit VBA-Code verbunden werden, so dass automatisch weitere Aktionen folgen können. **Ereignis**

In diesem Kapitel werden Ihnen zahlreiche Möglichkeiten von VBA vorgestellt, ohne in die Programmierung mit Variablen, Verzweigungen und Schleifen einsteigen zu müssen.

Im folgenden Kapitel werden dann diese wichtigen Elemente der Programmierung erläutert, die die automatisierte Bearbeitung von Excel erheblich verbessern.

2.1 Objekthierarchie und Auflistungen

Wie bereits erwähnt, verfügen Objekte über Eigenschaften. Eine Eigenschaft eines Objekts kann wiederum ein Unterobjekt sein, mit eigenen Eigenschaften, Methoden und Ereignissen. Setzt man diese Überlegung wei- **Hierarchie**

ter fort, so erhält man eine Hierarchie von Objekten, ausgehend von einem Hauptobjekt.

Application

Das Hauptobjekt bei Office-Anwendungen ist `Application` (engl. für Anwendung). In unserem Fall wird damit das Programm Excel selbst bezeichnet.

Workbooks

- Eine Eigenschaft des Objekts `Application` ist die Auflistung `Workbooks`. Darin befinden sich alle geöffneten Arbeitsmappen, also Excel-Dateien.

Worksheets

- Eine Eigenschaft einer einzelnen Arbeitsmappe ist die Auflistung `Worksheets`. Darin befinden sich alle Tabellenblätter einer Arbeitsmappe.

Range

- Eine Eigenschaft eines einzelnen Tabellenblatts ist das Objekt `Range`. Darin befinden sich Zellen und Zellbereiche eines Tabellenblatts.

Auflistung mit »s«

In Excel wird häufig mit Auflistungen gearbeitet, wie z. B. `Workbooks` oder `Worksheets`. Es handelt sich dabei um Sammlungen gleichartiger Objekte. Sie sind leicht an der Mehrzahlschreibweise (mit einem s am Ende) erkennbar.

2.2 Arbeitsmappen

Workbooks

Das Objekt `Workbooks` ist eine Auflistung. In dieser Auflistung befinden sich alle geöffneten Arbeitsmappen, also alle offenen Excel-Dateien.

Workbook

Das Objekt `Workbook` (ohne ein s am Ende) steht für eine einzelne Arbeitsmappe. Zur Bearbeitung einer bestimmten Arbeitsmappe gibt es verschiedene Möglichkeiten:

ThisWorkbook

- `ThisWorkbook`: Die Arbeitsmappe mit dem aktuell ausgeführten VBA-Code; sie wird im Folgenden auch als *diese* Arbeitsmappe bezeichnet.
- `ActiveWorkbook`: Die aktuell aktive Arbeitsmappe. Dies muss nicht *diese* Arbeitsmappe sein.
- `Workbooks(Index)`: `Index` ist die *laufende Nummer* der Arbeitsmappe innerhalb der `Workbooks`-Auflistung, von 1 bis Anzahl (Eigenschaft `Count`).

Workbooks("Name")

- `Workbooks("Name")`: Der Name der Arbeitsmappe als Zeichenkette (in Anführungszeichen)

2.2.1 Anzahl Arbeitsmappen ermitteln

Zählen

Mit folgender Prozedur wird die Anzahl der geöffneten Arbeitsmappen ermittelt:

```
Sub MappenZaehlen()
    MsgBox "Anzahl Mappen: " & Workbooks.Count
End Sub
```

Zur Erläuterung:

Count

- Es wird der Wert der Eigenschaft `Count` des Objekts `Workbooks` ermittelt und ausgegeben.
- Zu beachten ist, dass dabei die *Persönliche Makroarbeitsmappe* mitgezählt wird, falls sie vorhanden ist. Im *Visual Basic Editor* (*VBE*) können Sie sehen, ob es diese Mappe gibt.

Abbildung 2.1 Eigenschaft »Count«

2.2.2 Neue Arbeitsmappe erzeugen

Neue Mappe

Mit folgender Prozedur wird eine neue Mappe erzeugt und geöffnet. Zur Kontrolle wird die Anzahl der geöffneten Mappen vor und nach dem Erzeugen ausgegeben:

```
Sub NeueMappe()
    MsgBox Workbooks.Count
    Workbooks.Add
    MsgBox Workbooks.Count
End Sub
```

Zur Erläuterung:

Add()

- Es wird die Methode `Add()` des `Workbooks`-Objekts aufgerufen.
- Dadurch wird eine neue, leere Arbeitsmappe in Excel geöffnet. Diese ist dann die aktive Arbeitsmappe.
- Gleichzeitig wird der Auflistung `Workbooks` ein weiteres Element hinzugefügt, wie man an den beiden Ausgaben der Eigenschaft `Count` (vorher, nachher) erkennen kann.

2.2.3 Vorhandene Arbeitsmappe öffnen

Mappe öffnen

Mit folgender Prozedur wird eine vorhandene Mappe geöffnet:

```
Sub VorhandeneMappe()
    Workbooks.Open "C:\Temp\Mappe3.xlsm"
End Sub
```

Zur Erläuterung:

Open()

- Es wird die Methode `Open()` des `Workbooks`-Objekts aufgerufen. Dadurch wird die Arbeitsmappe mit dem angegebenen Namen im genannten Verzeichnis geöffnet. Diese ist dann die aktive Arbeitsmappe.
- Der Auflistung `Workbooks` wird ein weiteres Element hinzugefügt.
- Im Beispiel wurde eine Datei über eine absolute Pfadangabe angesprochen. Man kann auch Dateien mit relativen Pfadangaben und Dateien im gleichen Verzeichnis erreichen, siehe Abschnitt 2.2.10, »Pfad einer Arbeitsmappe ermitteln«.

Programmabbruch

- Falls die Arbeitsmappe (= Datei) nicht existiert, wird das Programm mit einer Fehlermeldung abgebrochen. Es kommt zu einem sogenannten Laufzeitfehler. In Kapitel 4, »Fehlerbehandlung«, werden Sie lernen, wie Sie solche Abbrüche vermeiden.

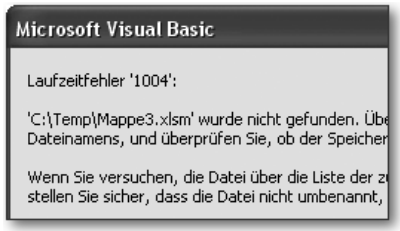


Abbildung 2.2 Programmabbruch durch Laufzeitfehler

2.2.4 Alle Arbeitsmappen schließen

Mappen schließen

Mit folgender Prozedur werden alle geöffneten Arbeitsmappen geschlossen:

```
Sub AlleMappenSchliessen()
    Workbooks.Close
End Sub
```

Zur Erläuterung:

- Es wird die Methode `Close()` des Objekts `Workbooks` aufgerufen. Sie schließt alle geöffneten Arbeitsmappen, aber nicht den VBE oder die Excel-Hilfe. **Close()**
- Die Anwendung Excel bleibt weiterhin geöffnet.
- Falls eine Arbeitsmappe geändert wurde, dann wird der Benutzer gefragt, ob er sie speichern möchte.

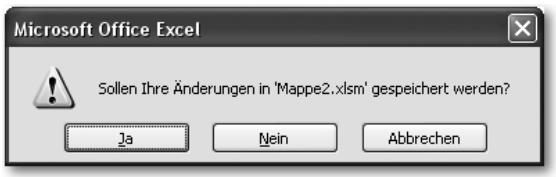


Abbildung 2.3 Nachfrage bei geänderter Datei

2.2.5 Name einer Arbeitsmappe ermitteln

Mit folgender Prozedur wird der Name einer Arbeitsmappe auf zwei Arten ermittelt: **Name der Mappe**

```
Sub MappenName()
    MsgBox "Name: " & ThisWorkbook.Name
    MsgBox "Name mit Pfad: " & ThisWorkbook.FullName
End Sub
```

Zur Erläuterung:

- `ThisWorkbook` verweist immer auf die Arbeitsmappe, in der diese Prozedur steht.
- Es wird der Wert der Eigenschaft `Name` ermittelt und ausgegeben, dies ist der Dateiname der Arbeitsmappe. **Eigenschaft »Name«**

Abbildung 2.4 Eigenschaft »Name«

- Zusätzlich wird der Wert der Eigenschaft `FullName` ermittelt und ausgegeben, dies ist der Dateiname der Arbeitsmappe inklusive vollständiger Pfadangabe. **FullName**

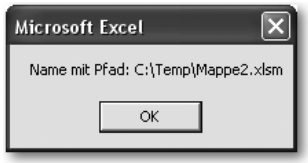


Abbildung 2.5 Eigenschaft »FullName«

2.2.6 Aktive Arbeitsmappe

Aktive Mappe

Mit folgender Prozedur wird in zwei verschiedenen Situationen der Name der aktiven Arbeitsmappe ermittelt:

```
Sub AktiveMappe()
    Workbooks.Open "C:\Temp\Mappe3.xlsm"
    MsgBox "Aktiv nach Öffnen: " & ActiveWorkbook.Name
    ActiveWorkbook.Close
    MsgBox "Aktiv nach Schließen: " & ActiveWorkbook.Name
End Sub
```

Zur Erläuterung:

- Zunächst wird eine weitere Arbeitsmappe geöffnet. Es wird in diesem Beispiel davon ausgegangen, dass die angegebene Datei existiert.

ActiveWorkbook

- `ActiveWorkbook` verweist immer auf die aktuell aktive Arbeitsmappe. Eine Arbeitsmappe ist immer dann aktiv, wenn der Benutzer sie ausgewählt oder der Entwickler sie per Programm aktiviert hat. Eine Arbeitsmappe, die soeben geöffnet wurde, ist automatisch aktiv.
- Zur Kontrolle wird der Name der aktiven Arbeitsmappe ausgegeben.

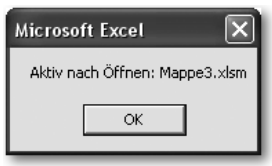


Abbildung 2.6 Aktive Arbeitsmappe nach dem Öffnen

- Es wird die Methode `Close()` des Objekts `Workbook` aufgerufen. Dadurch wird eine einzelne Arbeitsmappe, in diesem Falle die aktive Arbeitsmappe, geschlossen. Danach ist wieder die Arbeitsmappe aktiv, die vor dem Öffnen aktiv war.
- Zur Kontrolle wird wiederum der Name der nun aktiven Arbeitsmappe ausgegeben.

Abbildung 2.7 Wieder aktiv: »diese« Arbeitsmappe

2.2.7 Arbeitsmappe aktivieren

Mappe aktivieren

Mit folgender Prozedur wird eine bereits geöffnete Arbeitsmappe aktiviert. Dies ist dann sinnvoll, wenn mehrere Arbeitsmappen geöffnet sind und man sicher sein möchte, anschließend in einer bestimmten Arbeitsmappe weiterzuarbeiten.

```
Sub MappeAktivieren()
    ThisWorkbook.Activate
    MsgBox ActiveWorkbook.Name
End Sub
```

Zur Erläuterung:

- Es wird die Methode `Activate()` des Objekts `Workbook` aufgerufen. Dadurch wird die angesprochene Arbeitsmappe, in diesem Falle *diese* Arbeitsmappe, aktiviert. — Activate()
- Zur Kontrolle wird der Name der aktiven Arbeitsmappe ausgegeben.

2.2.8 Arbeitsmappe speichern

Mappe speichern

Mit folgender Prozedur wird diese Arbeitsmappe auf zwei verschiedene Arten gespeichert:

```
Sub MappeSichern()
    ThisWorkbook.Save
    ThisWorkbook.SaveAs "C:\Temp\Mappe2.xlsm"
    MsgBox "Gesichert: " & ThisWorkbook.Saved
End Sub
```

Zur Erläuterung:

- Es wird die Methode `Save()` des Objekts `Workbook` aufgerufen. Dadurch wird diese Arbeitsmappe gespeichert. — Save()
- Zusätzlich wird die Methode `SaveAs()` des Objekts `Workbook` aufgerufen. Dadurch wird diese Arbeitsmappe noch einmal gespeichert, und zwar im Verzeichnis *C:\Temp* unter dem Namen *Mappe2.xlsm*. An- — SaveAs()

schließend arbeiten Sie innerhalb der soeben angelegten Kopie der Mappe!

Saved

- Zur Kontrolle wird der Wert der Eigenschaft `Saved` ermittelt und ausgegeben. Dabei handelt es sich um einen sogenannten Wahrheitswert, also `False` (falsch) oder `True` (wahr). Da die Arbeitsmappe soeben gespeichert wurde, ist der Wert aktuell `True`. Nach einer Änderung in der Arbeitsmappe hätte die Eigenschaft den Wert `False`.

Abbildung 2.8 Eigenschaft »Saved«

Hinweis

Sie können den Wert der Eigenschaft `Saved` in einer Verzweigung nutzen. Je nach Wert der Eigenschaft (wahr oder falsch) kann bzw. sollte gespeichert werden.

2.2.9 Arbeitsmappe über Index oder Name auswählen

Mappe auswählen

Mit folgender Prozedur werden bereits geöffnete Arbeitsmappen nacheinander mit zwei verschiedenen Techniken aktiviert:

```
Sub MappeAusListe()
    Workbooks(1).Activate
    MsgBox ActiveWorkbook.Name
    Workbooks("Mappe2.xlsm").Activate
    MsgBox ActiveWorkbook.Name
End Sub
```

Zur Erläuterung:

Workbooks(Index)

- Zunächst wird eine bereits geöffnete Arbeitsmappe über den Index ausgewählt. Der Index ist die *laufende Nummer* der Arbeitsmappe innerhalb der `Workbooks`-Auflistung, von 1 bis Anzahl (Eigenschaft `Count`).
- Falls zwei Arbeitsmappen geöffnet sind, können nur die Indizes 1 oder 2 benutzt werden. Wenn Sie einen anderen Index wählen, erfolgt ein Abbruch mit einer Fehlermeldung. Wie bereits erwähnt,

werden Sie in Kapitel 4, »Fehlerbehandlung«, lernen, wie Sie solche Abbrüche vermeiden.

- Es ist nicht immer einfach, festzustellen, welches gerade die Arbeitsmappe Nummer 1, 2 usw. ist. Daher ist die folgende Methode vorzuziehen:
- Eine bereits geöffnete Arbeitsmappe wird über ihren Namen (in Anführungszeichen) ausgewählt. Dabei handelt es sich um den reinen Dateinamen, ohne Pfad. Da man in Excel nicht zwei Dateien mit dem gleichen Namen öffnen kann, auch wenn sie in unterschiedlichen Verzeichnissen stehen, ist der Name eindeutig.

Workbooks("Name")

2.2.10 Pfad einer Arbeitsmappe ermitteln

Pfad der Mappe

Häufig ist der Zugriff auf Arbeitsmappen im gleichen Verzeichnis oder einem Unterverzeichnis erforderlich. Dazu muss zunächst der Pfad dieser Arbeitsmappe (die den VBA-Code enthält) oder der Pfad der aktuellen Arbeitsmappe ermittelt werden. Ein Beispiel, in dem der Pfad von drei verschiedenen Arbeitsmappen ermittelt wird:

```
Sub PfadErmitteln()
    Workbooks.Open "C:\Temp\Mappe3.xlsm"
    MsgBox "Mappe3 ist in " & ActiveWorkbook.Path

    MsgBox "Diese Mappe ist in " & ThisWorkbook.Path
    Workbooks.Open ThisWorkbook.Path & "\Mappe1.xlsm"

    Workbooks.Open ThisWorkbook.Path & "\Zusatz\Test.xlsx"
    MsgBox "Test ist in " & ActiveWorkbook.Path
End Sub
```

Zur Erläuterung:

- Zunächst wird die Arbeitsmappe *C:\Temp\Mappe3.xlsm* geöffnet. Dies ist jetzt die aktive Arbeitsmappe.

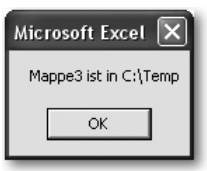


Abbildung 2.9 Pfad der Mappe »Mappe3«

Path

- Anschließend wird mit Hilfe der Eigenschaft `Path` der Pfad zu *dieser* Arbeitsmappe ausgegeben, in der sich die oben angegebene Prozedur `PfadErmitteln()` befindet.

Abbildung 2.10 Pfad von »dieser« Mappe

- Es wird die Arbeitsmappe *Mappe1.xlsm* geöffnet. Diese steht im gleichen Verzeichnis wie *diese* Arbeitsmappe, in der sich die oben angegebene Prozedur `PfadErmitteln()` befindet. Dies wird häufig benötigt, um sicher zu sein, dass eine Datei im gleichen Verzeichnis geöffnet wird.
- Zuletzt wird die Arbeitsmappe *Test.xlsx* geöffnet. Diese steht im Unterverzeichnis *Zusatz* des Verzeichnisses *dieser* Arbeitsmappe. Die soeben geöffnete Arbeitsmappe ist jetzt aktiv.

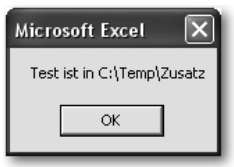


Abbildung 2.11 Pfad der Mappe »Test«

2.3 Tabellenblätter

Worksheets

Das Objekt `Worksheets` ist eine Auflistung, in der sich alle Tabellenblätter der jeweiligen Arbeitsmappe befinden.

Zur Bearbeitung eines einzelnen Tabellenblatts gibt es verschiedene Möglichkeiten:

- `ActiveSheet`: das aktuell aktive Arbeitsblatt
- `Worksheets(Index)`: `Index` ist die *laufende Nummer* des Tabellenblatts innerhalb der `Worksheets`-Auflistung, von 1 bis Anzahl (`Count`).

Worksheets("Name")

- `Worksheets("Name")`: der Name des Tabellenblatts als Zeichenkette (in Anführungszeichen)

Hinweis

In den folgenden Beispielen für den Einsatz von `Worksheets` wird jeweils vorher *diese* Arbeitsmappe aktiviert. Dies dient der eindeutigen Zuordnung für den Fall, dass mehrere Arbeitsmappen geöffnet sein sollten.

2.3.1 Tabellenblatt erzeugen

Neues Blatt

Mit folgender Prozedur wird in dieser Arbeitsmappe ein neues Tabellenblatt erzeugt:

```
Sub NeuesBlatt()
    ThisWorkbook.Activate
    MsgBox Worksheets.Count
    Worksheets.Add
    ActiveSheet.Name = "Neu"
    MsgBox Worksheets.Count
End Sub
```

Zur Erläuterung:

- Zunächst wird zur Kontrolle die Anzahl der Tabellenblätter dieser Arbeitsmappe (der Wert der Eigenschaft `Count` des Objekts `Worksheets`) ermittelt und ausgegeben.
- Es wird die Methode `Add()` des Objekts `Worksheets` aufgerufen. Dadurch wird ein neues Tabellenblatt vor dem aktiven Tabellenblatt erzeugt und eingefügt. Das neue Tabellenblatt wird dann automatisch zum aktiven Tabellenblatt. (Add())
- Durch die Angabe von Parametern könnten Sie genau bestimmen, an welcher Stelle das neue Tabellenblatt eingefügt werden soll, siehe nächster Abschnitt (»Tabellenblatt kopieren«).
- `ActiveSheet` bezeichnet das aktuell aktive Tabellenblatt. Der Name eines Tabellenblatts kann ermittelt bzw. geändert werden. (ActiveSheet)
- Es wird wiederum zur Kontrolle die Anzahl der Tabellenblätter ausgegeben. Sie hat sich erwartungsgemäß um 1 erhöht.

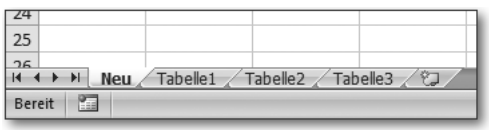


Abbildung 2.12 Neu erzeugtes und eingefügtes Tabellenblatt

2.3.2 Tabellenblatt kopieren

Blatt kopieren

Mit folgender Prozedur wird ein Tabellenblatt dieser Arbeitsmappe kopiert:

```
Sub BlattKopieren()
    ThisWorkbook.Activate
    Worksheets("Tabelle1").Copy After:=Worksheets("Tabelle3")
    ActiveSheet.Name = "Tab1Kopie"
End Sub
```

Zur Erläuterung:

Copy()

- Es wird die Methode `Copy()` des Objekts `Worksheets` aufgerufen. Dadurch wird ein Tabellenblatt kopiert. Es wird hinter einem Tabellenblatt in der gleichen Arbeitsmappe eingefügt, und es wird zum aktiven Tabellenblatt. Beide Tabellenblätter werden über ihren Namen angesprochen.

After, Before

- Bei `After` handelt es sich um einen benannten Parameter; diese können mit Hilfe des Operators := (Doppelpunkt Gleichheitszeichen) angegeben werden. Statt `After` ist auch `Before` möglich, dann wird das Tabellenblatt vor einem anderen Tabellenblatt in der gleichen Arbeitsmappe eingefügt.
- Falls gar kein Parameter angegeben wird, wird eine neue Arbeitsmappe erzeugt, die die Kopie enthält.
- Zum Abschluss wird noch der Name des Tabellenblatts geändert.

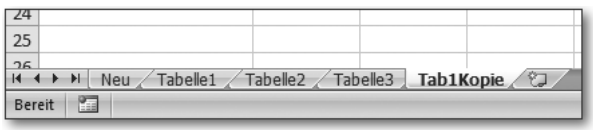


Abbildung 2.13 Kopiertes Tabellenblatt

2.3.3 Tabellenblatt verschieben

Blatt verschieben

Mit folgender Prozedur wird ein Tabellenblatt dieser Arbeitsmappe verschoben. Die zugehörige Methode `Move()` arbeitet sehr ähnlich wie die Methode `Copy()`:

```
Sub BlattVerschieben()
    ThisWorkbook.Activate
    Worksheets("Tab1Kopie").Move Before:=Worksheets("Tabelle1")
End Sub
```

Zur Erläuterung:

- Es wird die Methode `Move()` des Objekts `Worksheets` zum Verschieben eines Tabellenblatts aufgerufen. Es wird hinter einem Tabellenblatt in der gleichen Arbeitsmappe eingefügt, und es wird zum aktiven Tabellenblatt. **Move()**
- Statt mit `Before` hätten wir auch mit `After` oder ganz ohne Parameter arbeiten können, wie bei `Copy()`. **After, Before**

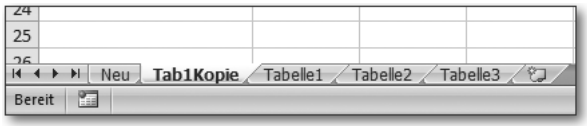


Abbildung 2.14 Verschobenes Tabellenblatt

2.3.4 Tabellenblatt löschen

Mit folgender Prozedur werden zwei Tabellenblätter dieser Arbeitsmappe gelöscht: **Blatt löschen**

```
Sub BlattLoeschen()
    ThisWorkbook.Activate
    Worksheets("Neu").Delete
    Worksheets("Tab1Kopie").Delete
End Sub
```

Zur Erläuterung:

- Es wird die Methode `Delete()` des Objekts `Worksheets` zum Löschen eines Tabellenblatts aufgerufen. Es erscheint jeweils eine Warnung, dass sich darin Daten befinden könnten. **Delete()**
- Beide Tabellenblätter werden über ihren Namen angesprochen.

Abbildung 2.15 Warnung beim Löschen eines Tabellenblatts

2.3.5 Tabellenblatt aktivieren

Blatt aktivieren

Mit folgender Prozedur werden nacheinander zwei Tabellenblätter dieser Arbeitsmappe aktiviert:

```
Sub BlattAktivieren()
    ThisWorkbook.Activate
    Worksheets("Tabelle3").Activate
    MsgBox ActiveSheet.Name
    Worksheets("Tabelle1").Activate
    MsgBox ActiveSheet.Name
End Sub
```

Zur Erläuterung:

Activate()

- Es wird die Methode `Activate()` des Objekts `Worksheets` zum Aktivieren eines Tabellenblatt aufgerufen. Dies ist dann sinnvoll, wenn Sie sicher sein möchten, anschließend in einem bestimmten Tabellenblatt weiterzuarbeiten. Aktionen in Zellen oder Bereichen beziehen sich anschließend auf dieses Tabellenblatt.
- Zur Kontrolle wird in der Prozedur der Name des jeweils aktiven Tabellenblatts ausgegeben.

2.3.6 Tabellenblatt formatieren

Cells

Die Eigenschaft `Cells` eines Tabellenblatts bietet die Möglichkeit, alle Zellen eines Tabellenblatts zu formatieren. Es können natürlich auch einzelne Zellen oder Zellbereiche formatiert werden, dazu mehr ab Abschnitt 2.4.7.

Alle Zellen

Nachfolgend werden alle Zellen des Tabellenblatts *Tabelle3* mit der Schriftart Arial, Schriftgröße 10 formatiert.

```
Sub BlattFormatieren()
    ThisWorkbook.Activate
    Worksheets("Tabelle3").Cells.Font.Name = "Arial"
    Worksheets("Tabelle3").Cells.Font.Size = 10
End Sub
```

Zur Erläuterung:

- Die Eigenschaft `Cells` bietet Zugriff auf alle Zellen eines Tabellenblatts.

Font

- Die Eigenschaft `Font` einer Zelle bestimmt die Schriftarteigenschaften.

- Die Untereigenschaften `Name` und `Size` dienen zur Festlegung von Name und Größe der Schriftart. Weitere Formatierungsmöglichkeiten erläutert Abschnitt 2.4.9, »Zellformat ›Schrift‹«.
- An dieser Stelle lässt sich wiederum gut die Objekthierarchie erkennen. Es wird die Untereigenschaft `Name` der Untereigenschaft `Font` der Eigenschaft `Cells` des Tabellenblatts geändert.

2.3.7 Gitternetz, Zeilen- und Spaltenüberschrift

Anzeigefenster gestalten

Sowohl Gitternetzlinien als auch Zeilen- und Spaltenüberschriften können ein- oder ausgeblendet werden. Dies sind eigentlich Eigenschaften des Anzeigefensters und nicht des Tabellenblatts. Sie sollen dennoch an dieser Stelle aufgeführt werden, da sie thematisch hierher gehören.

Eine Prozedur zum Ausblenden der genannten Objekte:

```
Sub BlattGitternetz()
    ThisWorkbook.Activate
    Worksheets("Tabelle3").Activate
    ActiveWindow.DisplayGridlines = False
    ActiveWindow.DisplayHeadings = False
End Sub
```

Zur Erläuterung:

- Zunächst wird das gewünschte Tabellenblatt aktiviert.
- **ActiveWindow** Das aktive Fenster (`ActiveWindow`) enthält nunmehr dieses Tabellenblatt.
- **DisplayGridlines** Die Eigenschaft `DisplayGridlines` bestimmt den Zustand: Gitternetzlinien ein/aus.
- **DisplayHeadings** Die Eigenschaft `DisplayHeadings` bestimmt den Zustand: Zeilen- und Spaltenüberschriften ein/aus.

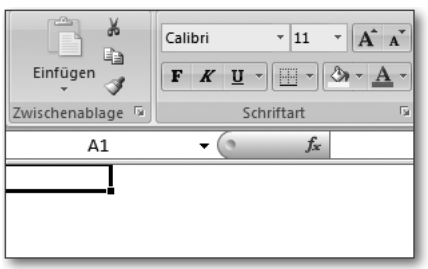


Abbildung 2.16 Tabellenblatt ohne Gitternetz und Überschriften

2.3.8 Seiteneinrichtung

PageSetup

Vorbereitungen für einen Ausdruck trifft man bei der Seiteneinrichtung. Für ein vorhandenes Tabellenblatt werden Sie die Seiteneinrichtung vom Tabellenblatt aus ausführen. Für ein neues Tabellenblatt, das erst von einer VBA-Anwendung mit Daten gefüllt wird, können Sie dazu die Eigenschaft `PageSetup` nutzen. In der folgenden Prozedur wird eine kleine Auswahl der zahlreichen Möglichkeiten getroffen.

Zeichen _ (Unterstrich)

Hinweis

Einige Anweisungen dieser Prozedur, aber auch noch vieler weiterer Prozeduren, sind sehr lang. Zeilen lassen sich mit Hilfe des Zeichens _ (Unterstrich) in mehrere übersichtliche Programmzeilen zerlegen, siehe Abschnitt 3.1.2, »Zeilen zerlegen«.

```
Sub BlattSeiteneinrichtung()
    ThisWorkbook.Activate

    ' Druckformat
    Worksheets("Tabelle3").PageSetup.Orientation = _
        xlLandscape

    ' Gitternetz, Zeilen- und Spaltenüberschrift
    Worksheets("Tabelle3").PageSetup.PrintGridlines = True
    Worksheets("Tabelle3").PageSetup.PrintHeadings = True

    ' Größenänderung
    Worksheets("Tabelle3").PageSetup.Zoom = False
    Worksheets("Tabelle3").PageSetup.FitToPagesWide = 1
    Worksheets("Tabelle3").PageSetup.FitToPagesTall = 99

    ' Horizontale Zentrierung
    Worksheets("Tabelle3").PageSetup.CenterHorizontally = _
        True

    ' Kopf- und Fußzeile
    Worksheets("Tabelle3").PageSetup.LeftHeader = _
        "Das ist die Kopfzeile"
    Worksheets("Tabelle3").PageSetup.CenterFooter = _
        "Seite &P von &N"
End Sub
```

Zur Erläuterung:

Orientation

- Die Eigenschaft `Orientation` bestimmt das Druckformat. Es gibt die Möglichkeiten `xlLandscape` (Querformat) und `xlPortrait` (Hochformat).

Druckeigenschaften

- Ähnlich wie beim Einrichten des Anzeigefensters wird mit den Eigenschaften `PrintGridlines` und `PrintHeadings` festgelegt, ob die Gitternetzlinien und die Zeilen- und Spaltenüberschriften ausgedruckt werden sollen oder nicht (`True` oder `False`).

Zoom

- Mit der Eigenschaft `Zoom` bestimmen Sie den Prozentsatz zur Vergrößerung oder Verkleinerung des Tabellenblatts für den Ausdruck. Falls dabei eine Zahl über 100 angegeben wird, wird das Tabellenblatt vergrößert gedruckt. Bei einer Zahl kleiner als 100 wird es verkleinert gedruckt. Der Wert `False` dient als Vorbereitung für eine Skalierung, die abhängig von der Seitenzahl ist.

FitToPages...

- Falls die Skalierung abhängig von der Seitenzahl ist, wird mit den Eigenschaften `FitToPagesWide` und `FitToPagesTall` jeweils eine Seitenanzahl festgelegt. Damit wird angegeben, auf wie vielen Seiten das Tabellenblatt in der Breite (`Wide`) und in der Höhe (`Tall`) ausgedruckt wird.

Center...

- Das Arbeitsblatt kann für den Ausdruck horizontal zentriert (`CenterHorizontally = True`), aber auch vertikal zentriert (`CenterVertically = True`) werden.

Header, Footer

- Den Text und die Anordnung von Kopf- und Fußzeile bestimmen die Eigenschaften `...Header` und `...Footer`. Es gibt jeweils die Präfixe `Left...`, `Right...` und `Center...` für die verschiedenen Bereiche. Im vorliegenden Beispiel wurden eine Kopfzeile links und eine Fußzeile zentriert angeordnet. Innerhalb des Texts, der als Zeichenkette zugewiesen wird, können bestimmte Variablen eingesetzt werden. Dabei gibt es u. a. die folgenden Möglichkeiten:
  - Seitennummer: `&P`
  - Seitenanzahl: `&N`
  - Druckdatum: `&D`
  - Druckuhrzeit: `&T`
  - Pfad zum Verzeichnis der Datei: `&Z`
  - Dateiname: `&F`
  - Tabellenblattname: `&A`

2.4 Zellen und Zellbereiche

Range, Cells

Mit Hilfe von `Range` bzw. `Cells` haben Sie zahlreiche Möglichkeiten, auf einzelne Zellen oder ganze Zellbereiche eines Tabellenblatts zuzugreifen. `Range` wurde bereits in Abschnitt 1.2, »Arbeiten mit Makros«, in den Beispielen zur Makroprogrammierung vorgestellt. Die aktuell aktive Zelle zur Bearbeitung wird mit `ActiveCell` bezeichnet.

Hinweis

In den folgenden Beispielen für den Einsatz von `Range` und `Cells` wird jeweils ein bestimmtes Tabellenblatt in *dieser* Arbeitsmappe aktiviert. Dies dient der eindeutigen Zuordnung für den Fall, dass vorher eine andere Arbeitsmappe oder ein anderes Tabellenblatt aktiviert sein sollte.

2.4.1 Zellen über »Range« auswählen

Zellbereich

Mit Hilfe von `Range` können sowohl zusammenhängende als auch nicht zusammenhängende Zellbereiche ausgewählt werden. Dabei werden ein Buchstabe für die Spalte und eine Nummer für die Zeile angegeben. Einige Möglichkeiten sehen Sie in Tabelle 2.1.

Range	Beschreibung
`Range("A3").Select`	einzelne Zelle
`Range("A3:F7").Select`	zusammenhängender Zellbereich
`Range("A3, C5, E2").Select`	mehrere nicht zusammenhängende Zellen
`Range("A8, B2:C4, E2").Select`	mehrere nicht zusammenhängende Zellen bzw. Zellbereiche

Tabelle 2.1 Range, Zellbereiche

Ganze Zeile oder Spalte

Bei den Zellbereichen kann es sich auch um ganze Spalten oder Zeilen handeln, wie Tabelle 2.2 zeigt.

Range	Beschreibung
`Range("A:A").Select`	ganze Spalte
`Range("C:E").Select`	mehrere zusammenhängende Spalten
`Range("B:D, F:F, H:I").Select`	mehrere nicht zusammenhängende Spalten

Tabelle 2.2 Range, ganze Spalten, ganze Zeilen

Range	Beschreibung
`Range("3:3").Select`	ganze Zeile
`Range("3:5").Select`	mehrere zusammenhängende Zeilen
`Range("3:5, 8:9, 12:12").Select`	mehrere nicht zusammenhängende Zeilen
`Range("A2:B4, 7:8, D:E, G2:H4").Select`	eine Mischung aus mehreren Möglichkeiten, siehe Abbildung 2.17

Tabelle 2.2 Range, ganze Spalten, ganze Zeilen (Forts.)

Beachten Sie, dass auch die beiden letzten Ausdrücke mit `Range` jeweils in eine Zeile gehören.

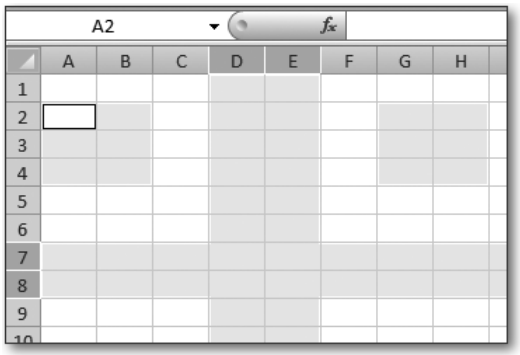


Abbildung 2.17 Range("A2:B4, 7:8, D:E, G2:H4").Select

Nicht zusammenhängender Bereich

Im letzten Beispiel wurden insgesamt vier nicht zusammenhängende Bereiche ausgewählt: zwei rechteckige Bereiche (A2:B4 und G2:H4), zwei ganze Spalten (D und E) und zwei ganze Zeilen (7 und 8). Die aktive Zelle (erkennbar am Rahmen) ist die erste ausgewählte Zelle (A2).

Mit folgender Prozedur werden Zellbereiche eines ausgewählten Tabellenblatts selektiert. Anschließend wird jeweils die Adresse der aktiven Zelle ausgegeben:

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

    Range("A2").Select
    MsgBox ActiveCell.Address
    Range("C4:G7").Select
    MsgBox ActiveCell.Address
    Range("A5, C3:G7").Select
```

```
    MsgBox ActiveCell.Address
    Range("C3:G7, A5").Select
    MsgBox ActiveCell.Address
    Range("C:D").Select
    MsgBox ActiveCell.Address
End Sub
```

Zur Erläuterung:

- Das Tabellenblatt *Tabelle1* dieser Arbeitsmappe wird aktiviert. Aktionen in Zellen oder Bereichen beziehen sich anschließend auf dieses Tabellenblatt.

Select()

- Es wird die Methode `Select()` des Objekts `Range` zum Auswählen von Zellen aufgerufen.

ActiveCell

- Eine der ausgewählten Zellen ist die aktive Zelle. Diese kann mit `ActiveCell` angesprochen werden. Falls mehrere Zellen ausgewählt wurden, ist die obere linke Zelle des ersten angegebenen Bereichs aktiv.

Address

- Allgemein ist `Address` die Adresse eines Zellbereichs. Hier geht es nur um die Adresse der aktiven Zelle.

Reihenfolge

- Nacheinander werden in diesem Beispiel ausgegeben: A2, C4, A5, C3, C1. Beachten Sie, dass im dritten und vierten Schritt die gleichen Zellen ausgewählt sind, aber sich die aktive Zelle aufgrund der Reihenfolge unterscheidet.

2.4.2 Zellen über »Cells« auswählen

Eine Zelle

`Cells` bietet nicht nur die Möglichkeit, alle Zellen eines Tabellenblatts zu erreichen, sondern auch einzelne Zellen oder Zellbereiche auszuwählen. Dabei werden eine Nummer für die Zeilen und eine Nummer für die Spalte angegeben.

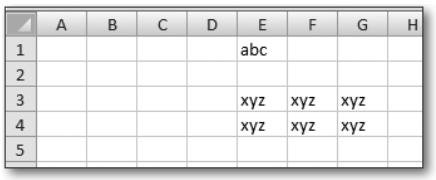
Das Arbeiten mit `Cells` bietet gegenüber dem Arbeiten mit `Range` Vorteile bei der Programmierung. Sowohl Zeilennummer als auch Spaltennummer können dann mit Hilfe von Variablen gebildet werden.

Mit folgender Prozedur werden Zellbereiche eines ausgewählten Tabellenblatts mit einem Wert versehen:

```
Sub ZellenMitCells()
    ThisWorkbook.Worksheets("Tabelle1").Activate
    Cells(1, 5).Value = "abc"
    Range(Cells(3, 5), Cells(4, 7)).Value = "xyz"
End Sub
```

Zur Erläuterung:

- Die Zelle `1, 5` bekommt mit Hilfe der Eigenschaft `Value` den Wert `abc`. Sie steht in Zeile 1, Spalte 5, ist also die Zelle E1.
- In Verbindung mit `Range` können auch Zellbereiche ausgewählt werden. Hier ist dies der Bereich von Zelle `3, 5` (= E3) bis Zelle `4, 7` (= G4). **Range mit Cells**



	A	B	C	D	E	F	G	H
1					abc			
2								
3					xyz	xyz	xyz	
4					xyz	xyz	xyz	
5								

Abbildung 2.18 Arbeiten mit »Cells«

2.4.3 Zellinhalte verschieben oder kopieren

Mit folgender Prozedur werden in einem ausgewählten Tabellenblatt:

- die Inhalte eines Zellbereichs verschoben
- die Inhalte eines anderen Zellbereichs kopiert

Beides haben Sie bereits im ersten Abschnitt mit Makros durchgeführt. Die hier vorgestellte Methoden umfassen weniger Codezeilen, sind besser zu warten und schneller im Ablauf: **Besserer Code**

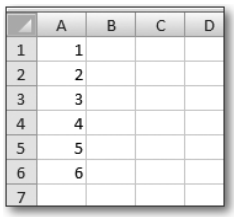
```
Sub ZellinhalteVerschiebenKopieren()
    ThisWorkbook.Worksheets("Tabelle1").Activate
    Range("A1:A2").Cut Destination:=Range("C1")
    Range("A5:A6").Copy Destination:=Range("C5")
End Sub
```

Zur Erläuterung:

- Es wird die Methode `Cut()` des Objekts `Range` zum Ausschneiden von Zellen aufgerufen. **Cut()**
  - Falls ein Ziel mit dem optionalen Parameter `Destination` angegeben ist, dann werden die Zellen dorthin verschoben. **Destination**
  - Falls kein Ziel angegeben ist, dann werden die Zellen in die Zwischenablage verschoben und können an anderer Stelle wieder eingefügt werden.
- Es wird die Methode `Copy()` des Objekts `Range` zum Kopieren von Zellen aufgerufen. Wie bei der Methode `Cut()` kann ein Ziel angege- **Copy()**

ben werden, ansonsten liegt die Kopie in der Zwischenablage und kann an anderer Stelle wieder eingefügt werden.

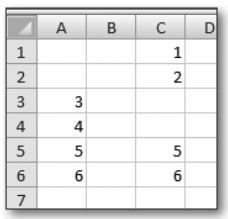
Vor dem Ablauf der Prozedur:



	A	B	C	D
1	1			
2	2			
3	3			
4	4			
5	5			
6	6			
7				

Abbildung 2.19 Originalinhalte

Nach dem Ablauf der Prozedur:



	A	B	C	D
1			1	
2			2	
3	3			
4	4			
5	5		5	
6	6		6	
7				

Abbildung 2.20 Nach dem Verschieben und Kopieren

2.4.4 Teile von Zellinhalten kopieren

Zwischenablage

Im vorherigen Abschnitt wurde bereits darauf hingewiesen, dass Zellinhalte auch in die Zwischenablage kopiert werden können. Dies ist von Vorteil, falls Sie nur ausgewählte Teile des Zellinhalts kopieren möchten, z. B. nur die Werte oder nur die Formate. Dies zeigt das folgende Beispiel:

```
Sub TeileKopieren()
    ThisWorkbook.Worksheets("Tabelle1").Activate
    Range("A1:A2").Copy
    Range("C1").PasteSpecial xlPasteValues
    Range("A5:A6").Copy
    Range("C5").PasteSpecial xlPasteFormats
End Sub
```

Zur Erläuterung:

- Es wird zweimal die Methode `Copy()` zum Kopieren eines Bereiches in die Zwischenablage aufgerufen.

- In beiden Fällen wird die Methode `PasteSpecial()` zum Einfügen spezieller Teile von Zellinhalten aufgerufen. Beim ersten Mal werden nur die Werte eingefügt, beim zweiten Mal nur die Formate. **PasteSpecial()**

Vor dem Ablauf der Prozedur:

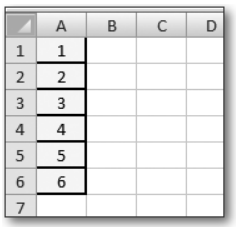


Abbildung 2.21 Original-Inhalte und -Formate

Nach dem Ablauf der Prozedur:

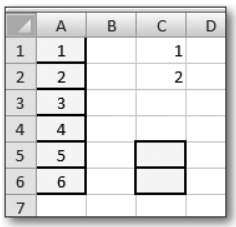


Abbildung 2.22 Nach dem Kopieren von Werten bzw. Formaten

2.4.5 Zellinhalt löschen

Mit folgender Prozedur werden bestimmte Zellinhalte gelöscht: **Löschen**

```
Sub ZellinhaltLoeschen()
    ThisWorkbook.Worksheets("Tabelle1").Activate
    Range("A1:A2").Clear
    Range("A3:A4").ClearContents
    Range("A5:A6").ClearFormats
End Sub
```

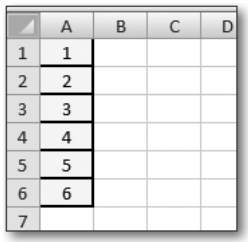
Zur Erläuterung:

- Die Methode `Clear()` des Objekts `Range` dient zum Löschen von Zellinhalten, inklusive Formatierung und Kommentaren. **Clear()**
- Mit Hilfe der Methode `ClearContents()` werden nur die Inhalte gelöscht. Formate und Kommentare bleiben erhalten. **ClearContents()**

ClearFormats()

- Die Methode `ClearFormats()` wird zum Löschen der Formate genutzt. Inhalte und Kommentare bleiben erhalten.

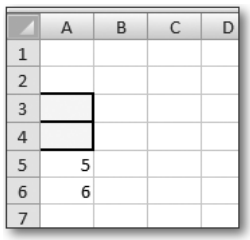
Vor dem Ablauf der Prozedur:



	A	B	C	D
1	1			
2	2			
3	3			
4	4			
5	5			
6	6			
7				

Abbildung 2.23 Original-Inhalte und -Formate

Nach dem Ablauf der Prozedur:



	A	B	C	D
1				
2				
3				
4				
5	5			
6	6			
7				

Abbildung 2.24 Nach dem Löschen von Werten und Formaten

2.4.6 Werte und Formeln eintragen

Zellinhalte eintragen

Mit folgender Prozedur werden Zahlen, Datumsangaben, Prozentzahlen und Formeln in Zellen eingetragen:

```
Sub WerteFormeln()
    ThisWorkbook.Worksheets("Tabelle2").Activate

    ' Zahl
    Range("A1").Value = 5.8
    Range("A2").Value = 1629.9
    Range("A3").FormulaLocal = "=SUMME(A1:A2)"

    ' Datum
    Range("A4").Value = "2008/03/01"
    Range("A5").Value = "2007/10/12"
    Range("A6").FormulaLocal = "=A4-A5"
```

```
    ' Zahl
    Range("A7").Value = 0.125
End Sub
```

Abbildung 2.25 zeigt das Ergebnis.

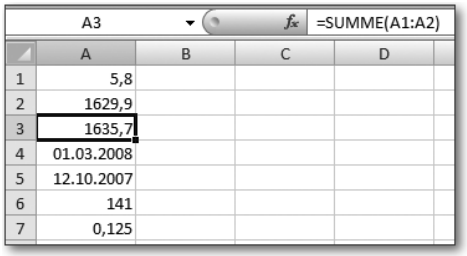


Abbildung 2.25 Werte und Formeln eintragen

Zur Erläuterung:

- Mit Hilfe der bereits bekannten Eigenschaft `Value` bekommen die Zellen A1 und A2 jeweils einen Zahlenwert zugewiesen. **Value**
- Bei Zahlen mit Nachkommastellen ist ein Punkt statt eines Kommas einzutragen. **Punkt statt Komma**
- Die Formel `=SUMME(A1:A2)` wird als Wert der Eigenschaft `FormulaLocal` in der von Excel gewohnten Form (in Deutsch) eingetragen. In der Bearbeitungszeile ist erkennbar, dass der Inhalt der Zelle A3 nach wie vor eine Formel ist. **FormulaLocal**
- Bei Datumsangaben dürfen die Anführungsstriche nicht fehlen. Es empfiehlt sich, die amerikanische Schreibweise zu benutzen, dann werden die Werte direkt als Datumsangaben erkannt und formatiert (führende Nullen). **Datum**
- Die Formel `=A4-A5` dient zur Berechnung der Tagesdifferenz zwischen den beiden Datumsangaben.
- Der Inhalt der Zelle A7 ist wiederum ein Zahlenwert.

2.4.7 Zellformat »Zahlen«

Mit folgender Prozedur werden die Formate für Zahlen, Datumsangaben und Prozentzahlen in Zellen eingetragen: **Format eintragen**

```
Sub ZellformatZahlen()
    ThisWorkbook.Worksheets("Tabelle2").Activate
    ' Zahl
    Range("A1:A3").NumberFormatLocal = "#.##0,00 €"
```

```
    ' Datum
    Range("A4:A5").NumberFormatLocal = _
        "TTTT, ""den"" TT. MMMM JJ"
    Range("A6").NumberFormatLocal = "0 ""Tage"""
    ' Prozent
    Range("A7").NumberFormatLocal = "0,00%"
End Sub
```

Das Ergebnis sehen Sie in Abbildung 2.26.

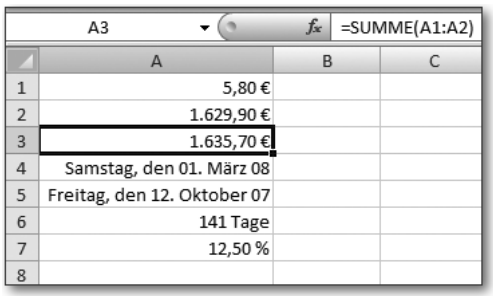


Abbildung 2.26 Zellformat Zahlen

Zur Erläuterung:

NumberFormatLocal

- Mit Hilfe der Eigenschaft `NumberFormatLocal` können Formate in der von Excel gewohnten Form (in Deutsch) eingetragen werden.

Formatierungszeichen

- Bei Zahlen, Währungs- oder Prozentangaben werden im Beispiel die folgende Zeichen verwendet:
  - `#`: Ziffer nur anzeigen, falls vorhanden
  - `0`: Ziffer immer anzeigen
  - , (Komma): Abtrennung der Nachkommastellen
  - . (Punkt): Tausenderpunkt
  - € (Euro): Währungszeichen
  - % (Prozent): Zahl wird durch 100 geteilt und mit einem Prozentzeichen dargestellt

Datumsformat

- Bei Datumsangaben werden im Beispiel die folgenden Zeichen verwendet:
  - `TTTT`: Wochentagsname in Deutsch, ausgeschrieben
  - `TT`: zwei Ziffern für den Tag im Monat
  - `MMMM`: Monatsname in Deutsch, ausgeschrieben
  - `JJ`: zwei Ziffern für das Jahr

- Eine Besonderheit ist zu beachten: Falls Text (z. B. die Wörter `den` oder `Tage`) in das Format integriert wird, so ist dieser bekanntlich in Excel in doppelten Anführungsstrichen zu notieren. So kann kein Konflikt mit den festen Formatangaben auftreten. Da der Wert der Eigenschaft `NumberFormatLocal` schon in doppelten Anführungszeichen steht, muss dieser Formatierungstext in zweifachen doppelten Anführungsstrichen stehen. **Text im Format**

2.4.8 Zellformat »Ausrichtung«

Ausrichten

Mit folgender Prozedur werden Zellen auf verschiedene Art und Weise ausgerichtet:

```
Sub ZellformatAusrichtung()
    ThisWorkbook.Worksheets("Tabelle2").Activate

    Range("C1").Value = "Hallo"
    Range("C1").HorizontalAlignment = xlCenter
    Range("C1").VerticalAlignment = xlTop

    Range("C2").Value = "Das ist ein längerer Text"
    Range("C2").WrapText = True

    Range("C3").Value = "Hallo"
    Range("C3:C7").MergeCells = True
    Range("C3:C7").Orientation = xlVertical

    Range("C8").Value = "Hallo"
    Range("C8").Orientation = 45
End Sub
```

Das Ergebnis ist in Abbildung 2.27 dargestellt.

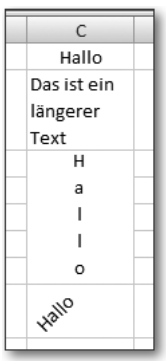


Abbildung 2.27 Zellformat »Ausrichtung«

Zur Erläuterung:

...Alignment

- Text wird mit Hilfe der Eigenschaft `HorizontalAlignment` und `VerticalAlignment` ausgerichtet. Zulässige Konstanten für
  - die horizontale Ausrichtung sind z. B. `xlLeft` (linksbündig), `xlRight` (rechtsbündig), `xlCenter` (zentriert) und `xlJustify` (Blocksatz);
  - die vertikale Ausrichtung sind z. B. `xlBottom` (am unteren Rand), `xlCenter` (vertikal zentriert) und `xlTop` (am oberen Rand).

WrapText

- Die Eigenschaft `WrapText` bestimmt die Anordnung längerer Texte:
  - Falls `WrapText` den Wert `True` hat, wird der Text innerhalb der Zelle über mehrere Zeilen verteilt.
  - Falls `WrapText` den Wert `False` hat, wird der Text nur in einer Zeile geschrieben, die gegebenenfalls über den rechten Rand der Zelle hinausgeht.

MergeCells

- Die Eigenschaft `MergeCells` bestimmt darüber, ob Zellen verbunden werden können:
  - Wenn `MergeCells` den Wert `True` hat, werden die Zellen des genannten Bereichs miteinander verbunden dargestellt.
  - Wenn `MergeCells` den Wert `False` hat, werden die Zellen des genannten Bereichs einzeln dargestellt.

Orientation

- Die einzelnen Zeichen eines Zellinhalts können mit Hilfe der Eigenschaft `Orientation`
  - übereinander geschrieben werden, mit dem Wert `xlVertical`;
  - in einem schrägen Winkel geschrieben werden, mit einem Zahlenwert zwischen –90 Grad und +90 Grad. Der Normalwert ist 0 Grad.

2.4.9 Zellformat »Schrift«

Schrift

Mit folgender Prozedur werden die Schrifteigenschaften von Zellen bestimmt:

```
Sub ZellformatSchrift()
    ThisWorkbook.Worksheets("Tabelle2").Activate
    Range("C1").Font.Name = "Tahoma"
    Range("C1").Font.Bold = True
    Range("C1").Font.Italic = True
    Range("C1").Font.Underline = True
    Range("C1").Font.Size = 20
    Range("C1").Font.Color = vbRed
End Sub
```

Das Ergebnis:

Abbildung 2.28 Zellformat »Schrift«

Zur Erläuterung:

- Die Eigenschaft `Font` steht für die Formatierung der Schriftart in der Zelle. Sie hat zahlreiche Untereigenschaften, z. B.: **Font**
  - `Name` für den Namen der Schriftart
  - `Bold` für fett (`True` oder `False`)
  - `Italic` für kursiv (`True` oder `False`)
  - `Underline` für unterstrichen (`True` oder `False`)
  - `Size` für die Größe der Schrift
  - `Color` für die Farbe der Schrift
- Farben können auf verschiedene Art und Weise angegeben werden, z. B.: **Farbe**
  - Speziell für die Grundfarben mit Hilfe von Farbkonstanten, siehe Abschnitt 3.2.4, »Konstanten«.
  - Allgemein mit der Funktion `RGB()`. Diese Funktion hat drei Parameter, die für Rot-, Grün- und Blau-Werte der Farbe stehen, jeweils zwischen 0 und 255. Im vorliegenden Beispiel hätte also die folgende Anweisung dasselbe bewirkt: `Range("C1").Font.Color = RGB(255, 0, 0)` **RGB()**

2.4.10 Einzelne Zeichen formatieren

Mit folgender Prozedur werden einzelne Zeichen innerhalb einer Zeichenkette formatiert: **Zeichenformatierung**

```
Sub EinzelneZeichen()
    ThisWorkbook.Worksheets("Tabelle2").Activate

    Range("E3").Value = "x2"
    Range("E3").Characters(2, 1).Font.Superscript = True

    Range("E4:E6").Value = "a38 + a39"
    Range("E4:E6").Characters(2, 2).Font.Subscript = True
    Range("E4:E6").Characters(8, 2).Font.Subscript = True
End Sub
```

Das Ergebnis:

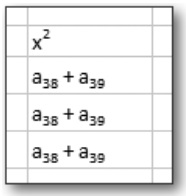


Abbildung 2.29 Einzelne Zeichen formatieren

Zur Erläuterung:

Characters

- Die Eigenschaft `Characters` liefert Teile einer Zeichenfolge. Diese können anschließend separat formatiert werden. Es müssen zwei Werte festgelegt werden:
  - Die erste Ziffer bestimmt die Stelle, an der die Teilzeichenfolge beginnt. Das erste Zeichen einer Zeichenfolge hat die Nummer 1.
  - Die zweite Ziffer bestimmt die Länge der Teilzeichenfolge.

Superscript

- Im ersten Beispiel wird die Zeichenfolge `2` geliefert. Mit Hilfe der Untereigenschaft `Superscript` der `Font`-Eigenschaft des Textes wird dieses Zeichen hochgestellt.

Subscript

- Im zweiten Beispiel werden für insgesamt drei Zellen jeweils die beiden Zeichenfolgen `38` und `39` geliefert. Mit Hilfe der Untereigenschaft `Subscript` der `Font`-Eigenschaft des Textes wird dieses Zeichen tiefgestellt.

2.4.11 Zellformat »Rahmen«

Rahmen

Mit folgender Prozedur werden Zellen ganz oder teilweise eingerahmt:

```
Sub ZellformatRahmen()
    ThisWorkbook.Worksheets("Tabelle2").Activate

    Range("E3").Borders.LineStyle = xlDouble
    Range("E3").Borders.Weight = xlThick
    Range("E3").Borders.Color = vbGreen

    Range("E4:E6").Borders(xlEdgeLeft).Weight = xlThin
    Range("E4:E6").Borders(xlEdgeRight).Weight = xlThin
    Range("E4:E6").Borders(xlInsideHorizontal).Weight = _
        xlHairline
End Sub
```

Das Ergebnis:

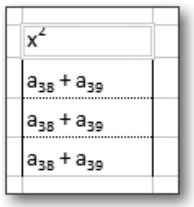


Abbildung 2.30 Zellformat »Rahmen«

Zur Erläuterung:

Borders

- Die Eigenschaft `Borders` dient zur Bearbeitung des Rahmens.
  - Falls nach `Borders` keine weitere Angabe in Klammern erfolgt, werden Eigenschaften für den gesamten Rahmen eingestellt.

Rahmenteile

  - Es können aber auch die Eigenschaften einzelner Teile des Rahmens bestimmt werden. Zulässige Konstanten sind z. B.: `xlEdgeLeft` (linker Rahmen), `xlEdgeRight` (rechter Rahmen), `xlEdgeTop` (oberer Rahmen), `xlEdgeBottom` (unterer Rahmen), `xlInsideHorizontal` (innere horizontale Zwischenlinien), `xlInsideVertical` (innere vertikale Zwischenlinien).

LineStyle

- Mit der Eigenschaft `LineStyle` wird die Linienart für den Rahmen festgelegt. Zulässige Konstanten sind z. B.: `xlContinuous` (durchgehende Linie), `xlDot` (gepunktete Linie) und `xlDouble` (doppelte Linie).

Weight

- Die Eigenschaft `Weight` bestimmt die Dicke des Rahmens. Zulässige Konstanten sind: `xlHairline` (ganz dünne Linie), `xlThin` (dünne Linie), `xlMedium` (mitteldicke Linie) und `xlThick` (dicke Linie).

Color

- Durch die Eigenschaft `Color` wird die Rahmenfarbe bestimmt. Wie bei allen Farbangaben kann mit den Farbkonstanten oder mit der Funktion `RGB()` gearbeitet werden.

2.4.12 Zellformat »Muster«

Muster

Mit folgender Prozedur wird das Hintergrundmuster von Zellen gestaltet:

```
Sub ZellformatMuster()
    ThisWorkbook.Worksheets("Tabelle2").Activate
    Range("E3, E6").Interior.Color = vbYellow
End Sub
```

Das Ergebnis:

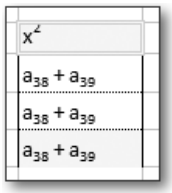


Abbildung 2.31 Zellformat »Muster«

Zur Erläuterung:

Interior

- Die Eigenschaft `Interior` dient zur Bearbeitung des Innenbereichs einer Zelle.
- Sie hat verschiedene Untereigenschaften, u. a. `Color` für die Innenfarbe.

2.4.13 Zellen einfügen

Zellbereiche einfügen

Mit folgender Prozedur werden Zellbereiche in einem ausgewählten Tabellenblatt eingefügt:

```
Sub ZelleEinfuegen()
    ThisWorkbook.Worksheets("Tabelle1").Activate
    Range("A2:A3").Insert Shift:=xlShiftDown
    Range("6:7").Insert
End Sub
```

Zur Erläuterung:

Insert()

- Es wird die Methode `Insert()` des Objekts `Range` zum Einfügen von Zellen aufgerufen.

Shift

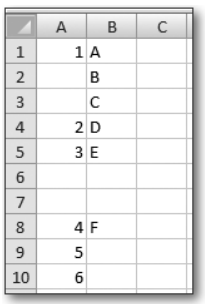
- Mit Hilfe des optionalen Parameters `Shift` entscheiden Sie, was mit den Nachbarzellen passieren soll. Falls dieser Parameter weggelassen wird, entscheidet Excel anhand der Bereichsform.
- Im ersten Fall werden die unteren Nachbarzellen nach unten verschoben. Dazu dient die Konstante `xlShiftDown`. Die Konstante `xlShiftToRight` hätte die rechten Nachbarzellen nach rechts verschoben.
- Im zweiten Fall sind ganze Zeilen ausgewählt. Es kann nur eine Verschiebung nach unten stattfinden, daher macht die Angabe des Parameters `Shift` keinen Sinn.

Vor dem Ablauf der Prozedur:

	A	B	C
1	1	A	
2	2	B	
3	3	C	
4	4	D	
5	5	E	
6	6	F	
7			
8			
9			
10			

Abbildung 2.32 Original-Struktur der Tabelle

Nach dem Ablauf der Prozedur:



	A	B	C
1	1	A	
2		B	
3		C	
4	2	D	
5	3	E	
6			
7			
8	4	F	
9	5		
10	6		

Abbildung 2.33 Nach dem Einfügen von Zellen und Zeilen

Hinweis

EntireRow, EntireColumn

Die Anweisung `Range("A2:A3").EntireRow.Insert` würde zwei ganze neue Zeilen vor den Zeilen des angegebenen Bereichs einfügen, hier also vor den Zeilen 2 und 3. Entsprechend würde durch die Anweisung `Range("A2:A3").EntireColumn.Insert` eine ganze neue Spalte vor der Spalte A eingefügt.

2.4.14 Zellen löschen

Löschen

Mit folgender Prozedur werden Zellbereiche eines ausgewählten Tabellenblatts gelöscht:

```
Sub ZelleLoeschen()
    ThisWorkbook.Worksheets("Tabelle1").Activate
    Range("6:7").Delete
    Range("A2:A3").Delete Shift:=xlShiftUp
End Sub
```

Zur Erläuterung:

Delete()
- Es wird die Methode `Delete()` des Objekts `Range` zum Löschen von Zellen aufgerufen.

Shift
- Auch hier gibt es einen optionalen Parameter `Shift`, mit dem Sie entscheiden, was mit den Nachbarzellen passieren soll. Falls dieser Parameter weggelassen wird, entscheidet Excel wiederum aufgrund der Bereichsform.
- Im ersten Fall sind ganze Zeilen ausgewählt. Es kann nur eine Verschiebung (aller Zeilen darunter) nach oben stattfinden, daher ergibt die Angabe des Parameters `Shift` keinen Sinn.
- Im zweiten Fall werden die unteren Nachbarzellen nach oben verschoben. Dazu dient die Konstante `xlShiftUp`. Die Konstante `xlShiftToLeft` hätte die rechten Nachbarzellen nach links verschoben.

Im vorliegenden Beispiel wird durch das Löschen von Zellen und Zeilen das vorherige Einfügen von Zellen und Zeilen rückgängig gemacht.

EntireRow, EntireColumn

Hinweis

Die Anweisung `Range("A2:A3").EntireRow.Delete` würde die gesamten Zeilen des angegebenen Bereichs löschen, hier also die Zeilen 2 und 3. Entsprechend würde durch die Anweisung `Range("A2:A3").EntireColumn.Delete` die gesamte Spalte A gelöscht.

2.4.15 Zeilenhöhe und Spaltenbreite

Höhe, Breite

Mit folgender Prozedur werden die Höhe bestimmter Zeilen und die Breite bestimmter Spalten eingestellt:

```
Sub HoeheBreite()
    ThisWorkbook.Worksheets("Tabelle2").Activate
    Range("1:2").RowHeight = 55
    Range("B:B,D:D").ColumnWidth = 3
End Sub
```

Das Ergebnis ist in Abbildung 2.34 dargestellt.

Zur Erläuterung:

- Als Zellbereich werden die Zeilen 1 und 2 sowie die Spalten B und D ausgewählt.

RowHeight
- Die Eigenschaft `RowHeight` verändert die Zeilenhöhe.

ColumnWidth
- Die Eigenschaft `ColumnWidth` bestimmt die Spaltenbreite.

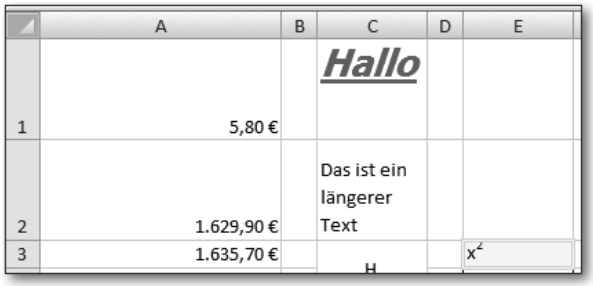


Abbildung 2.34 Zeilenhöhe und Spaltenbreite

Hinweis

AutoFit, Columns, Rows

Die optimale Einstellung der Zeilenhöhe und der Spaltenbreite lässt sich über die Methode `AutoFit()` des Unterobjekts `Columns` bzw. `Rows` erreichen. Ein Beispiel: `Range("G:G").Columns.AutoFit` stellt die Breite der Spalte G optimal ein. `Range("1:2").Rows.AutoFit` stellt die Höhe der Zeilen 1 und 2 optimal ein.

2.4.16 Benutzten Zellbereich erkennen

Benutzter Bereich

Eine nützliche Eigenschaft eines Tabellenblatts ist der *benutzte Bereich* (`UsedRange`). Damit ist der kleinste zusammenhängende rechteckige Zellbereich gemeint, der alle nicht leeren Zellen umfasst.

Mit folgender Prozedur werden alle Zellen dieses Bereichs durch einen roten Rahmen hervorgehoben und gezählt:

```
Sub BenutzterBereich()
    ThisWorkbook.Worksheets("Tabelle1").Activate
    ActiveSheet.UsedRange.Borders.Color = vbRed
    ActiveSheet.UsedRange.Interior.Color = vbYellow
    MsgBox "Anzahl: " & ActiveSheet.UsedRange.Count
End Sub
```

Nehmen wir an, folgende Zellen wurden benutzt:

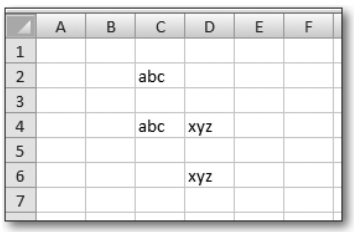


Abbildung 2.35 Zellen mit Inhalt

Der benutzte Bereich ergibt sich dann wie folgt:

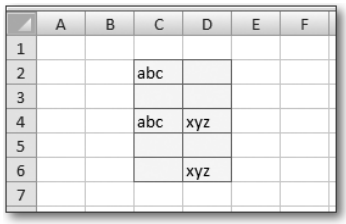


Abbildung 2.36 Benutzter Bereich

Zur Erläuterung:

UsedRange

- Mit `ActiveSheet.UsedRange` werden alle benutzten Zellen des aktiven Tabellenblatts erfasst.
- Die Eigenschaft `Count` ergibt für einen `Range`, wie bei Arbeitsmappen oder Tabellenblättern, eine Anzahl. In diesem Falle beträgt die Anzahl 10 Zellen.

2.4.17 Spezielle Zellen erkennen

SpecialCells

Die Methode `SpecialCells` ist in der Lage, spezielle Zellen zu erkennen. Anschließend können diese Zellen besonders bearbeitet bzw. hervorgehoben werden. Im folgenden Beispiel werden farblich hervorgehoben bzw. gezählt:

- alle Zellen eines Bereichs, die eine Formel enthalten
- alle Zellen eines Bereichs, die leer sind
- die letzte Zelle eines Bereichs, in diesem Falles des benutzten Bereichs

```
Sub SpezielleZellen()
    ThisWorkbook.Worksheets("Tabelle2").Activate

    'Zellen mit Formeln
    Range("A1:A8").SpecialCells(xlCellTypeFormulas). _
        Interior.Color = vbYellow
    MsgBox "Formeln: " & Range("A1:A9"). _
        SpecialCells(xlCellTypeFormulas).Count

    'Leere Zellen
    Range("A1:A8").SpecialCells(xlCellTypeBlanks). _
        Interior.Color = vbCyan
    MsgBox "Leer: " & Range("A1:A8").SpecialCells _
        (xlCellTypeBlanks).Count
```

```
    'Letzte benutzte Zelle
    ActiveSheet.UsedRange.SpecialCells _
        (xlCellTypeLastCell).Interior.Color = vbGreen
    MsgBox "Letzte Zeile: " & ActiveSheet.UsedRange. _
        SpecialCells(xlCellTypeLastCell).Row
    MsgBox "Letzte Spalte: " & ActiveSheet.UsedRange. _
        SpecialCells(xlCellTypeLastCell).Column
End Sub
```

Abbildung 2.37 zeigt das Ergebnis.

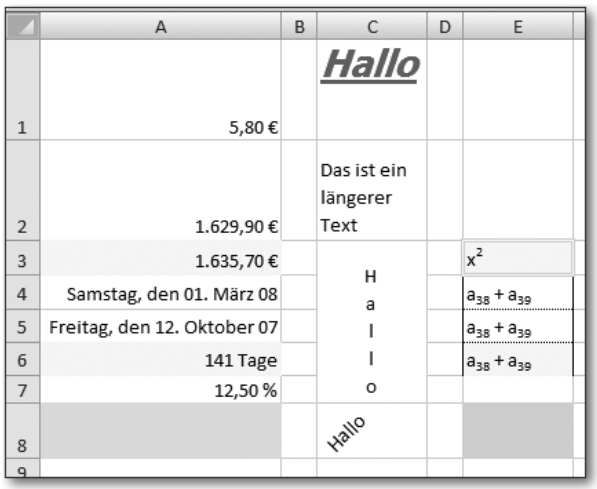


Abbildung 2.37 Spezielle Zellen

Zur Erläuterung:

- Die Methode `SpecialCells` bekommt einen bestimmten Zelltyp in Form einer Konstante übergeben. **SpecialCells**
- Der Typ `xlCellTypeFormulas` liefert Zellen, die Formeln enthalten. Hier sind dies die Zellen A3 und A6. **Zelle mit Formel**
- Der Typ `xlCellTypeBlanks` liefert Zellen, die leer sind, hier die Zelle A8. **Leere Zelle**
- Der Typ `xlCellTypeLastCell` liefert die letzte Zelle eines Bereichs. In diesem Falle ist es der benutzte Bereich (`UsedRange`), es handelt sich um die Zelle E8. **Letzte Zelle**
- Mit Hilfe der Eigenschaften `Row` und `Column` können Sie sich die Zeilennummer und die Spaltennummer einer Zelle angeben lassen. In diesem Falle ist es die Zeile und die Spalte der letzten Zelle des benutzten Bereichs.

Hinweis

Bei einem Zellbereich wird mit den Eigenschaften `Row` bzw. `Column` die Nummer der Zeile bzw. der Spalte der aktiven Zelle ausgegeben. Ein Beispiel: `Range("A3:B5").Row` ergibt 3, `Range("A3:B5").Column` ergibt 1.

2.4.18 Versatz mit Offset

Offset

Mit dem *Offset* für einen Zellbereich bezeichnet man einen zweiten Zellbereich, der gegenüber dem Ursprungsbereich versetzt ist. Dieser Bereich ist wiederum ein `Range`-Objekt. Dieser Versatz kann sowohl in Bezug auf einzelne Zellen als auch in Bezug auf zusammenhängende oder nicht zusammenhängende Zellbereiche angewendet werden. Hier ein Beispiel:

```
Sub ZellOffset()
    ThisWorkbook.Worksheets("Tabelle3").Activate

    ' Offset berechnen
    Cells(4, 3).Value = "Ber 1"
    Cells(4, 3).Offset(0, 2).Value = "Off(0,2)"
    Cells(4, 3).Offset(-3, 1).Value = "Off(-3,1)"
    Cells(4, 3).Offset(-2, -1).Value = "Off(-2,-1)"

    ' Offset eines Bereichs
    Range("C6:D8,E9").Value = "Ber 2"
    Range("C6:D8,E9").Offset(4, 0).Value = "Off(4,0)"

    ' Offset-relative Zellen
    Cells(4, 3).Offset(0, 2).Range("A2").Value = "A2"
    Cells(4, 3).Offset(0, 2).Range("A3").Value = "A3"
    Cells(4, 3).Offset(0, 2).Range("B1").Value = "B1"
    Cells(4, 3).Offset(0, 2).Range("B2").Value = "B2"
    Cells(4, 3).Offset(0, 2).Range("B3").Value = "B3"
End Sub
```

Offset berechnen:

- Ausgangszelle ist die Zelle `(4, 3)` oder anders bezeichnet: C4.

Zeile, Spalte

- Bei einem Offset wird zuerst der Versatz für die Zeile, dann für die Spalte angegeben.
- Falls der Ursprungsbereich mit `Cells` angegeben ist, können Sie den versetzten Bereich leicht ermitteln, indem Sie die Zahlen für die Zeile und die Spalte jeweils addieren.

- Ein Offset kann auch negativ sein, dann handelt es sich um Zeilen oberhalb oder um Spalten links vom Ursprungsbereich. **Negativer Offset**
- Die drei Offsetangaben für den ersten Bereich ergeben:
  - Zelle 4,3 + Offset 0,2 = Zelle 4,5 = Zelle E4
  - Zelle 4,3 + Offset 3,1 = Zelle 1,4 = Zelle D1
  - Zelle 4,3 + Offset 2,1 = Zelle 2,2 = Zelle B2

Das Ergebnis:

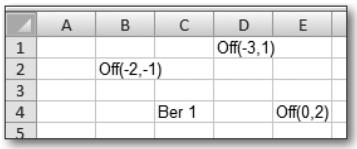


Abbildung 2.38 Offset berechnen

Offset eines Bereichs:

- Die Offsetangabe für den nicht zusammenhängenden Zellbereich ergibt einen gleichartigen Bereich, der um vier Zeilen (`Offset(4, 0)`) nach unten verschoben ist.

Abbildung 2.39 zeigt das Ergebnis.

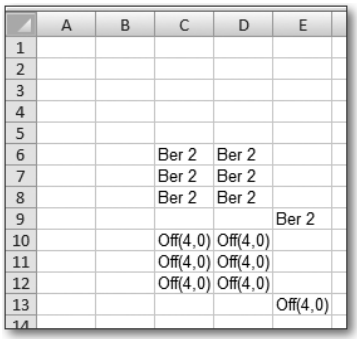


Abbildung 2.39 Offset eines Bereichs

Offset-relative Zellen:

- Die erste Zelle eines Offsets, egal ob einzelne Zelle oder Zellbereich, wird relativ mit A1 bezeichnet. So können Sie jede Zelle im Offset eindeutig erreichen. **Offset-relative Bezeichnung**
- Im Beispiel sind die Zellen A2, A3, B1, B2 und B3 des Offsets besonders hervorgehoben.

Das Ergebnis:

	A	B	C	D	E	F
1						
2						
3						
4			Ber 1		Off(0,2)	B1
5					A2	B2
6					A3	B3
7						

Abbildung 2.40 Offset-relative Zellen

2.4.19 Zellbereich sortieren

Sortieren

Die Methode `Sort()` des `Range`-Objekts bietet zahlreiche Möglichkeiten, Zellbereiche zu sortieren. Ausgangspunkt soll die folgende Tabelle sein:

	A	B	C
1	Nachname	Vorname	Nummer
2	Maier	Herbert	23
3	Maier	Julia	15
4	Schmitz	Peter	18
5			

Abbildung 2.41 Tabelle, die sortiert werden soll

Diese Tabelle soll auf zwei Arten sortiert werden:

- nach Nummer
- nach Nachname und Vorname

Die oberste Zeile soll als Überschrift erkannt werden, so dass sie nicht einsortiert wird.

Zunächst die Prozedur für die Sortierung nach Nummer:

```
Sub SortierenNummer()
    ThisWorkbook.Worksheets("Tabelle4").Activate
    Range("A1:C4").Sort Key1:=Range("C1:C4"), Header:=xlYes
End Sub
```

Das Ergebnis sehen Sie in Abbildung 2.42.

	A	B	C
1	Nachname	Vorname	Nummer
2	Maier	Julia	15
3	Schmitz	Peter	18
4	Maier	Herbert	23
5			

Abbildung 2.42 Sortierung nach Nummer

Zur Erläuterung:

- Es wird der angegebene Bereich (A1:C4) sortiert.
- Die Methode `Sort()` hat zahlreiche Parameter, daher werden die Parameter beim Aufruf benannt. **Sort()**
- Der Parameter `Key1` gibt den Bereich an, in dem der Sortierschlüssel steht. Hier ist dies die dritte Spalte mit der Nummer, also C1:C4. **Key1**
- Der Parameter `Order1` bestimmt darüber, ob aufsteigend (`xlAscending`) oder absteigend (`xlDescending`) sortiert wird. Da aufsteigende Sortierung der Standard ist, muss der Parameter nicht angegeben werden. **Order1**
- Über den Parameter `Header` legen Sie fest, ob die oberste Zeile des Bereichs als Überschrift erkannt werden soll. Der Standard ist `xlNo`, daher wird hier `xlYes` angegeben. Falls der Wert `xlGuess` angegeben wird, entscheidet Excel, ob eine Überschrift vorhanden ist, und reagiert entsprechend. **Header**

Es folgt die Prozedur für die Sortierung nach Nachname und Vorname:

```
Sub SortierenName()
    ThisWorkbook.Worksheets("Tabelle4").Activate
    Range("A1:C4").Sort Key1:=Range("A1:A4"), _
        Key2:=Range("B1:B4"), Header:=xlYes
End Sub
```

Abbildung 2.43 zeigt das Ergebnis.

	A	B	C
1	Nachname	Vorname	Nummer
2	Maier	Herbert	23
3	Maier	Julia	15
4	Schmitz	Peter	18
5			

Abbildung 2.43 Sortierung nach Nachname und Vorname

Zur Erläuterung:

- Der Parameter `Key2` gibt den Bereich an, in dem der zweite Sortierschlüssel steht. Es wird also zunächst nach der ersten Spalte mit den Nachnamen sortiert, bei gleichem Nachnamen nach der zweiten Spalte mit den Vornamen. **Key2**
- Es können bis zu drei Sortierschlüssel (`Key1` bis `Key3`) angegeben werden. Auf- bzw. absteigende Sortierung wird über die Parameter `Order1` bis `Order3` bestimmt.

2.5 Ereignisprozeduren

Ereignis passiert

Ereignisse sagen etwas darüber aus, was gerade mit einem Objekt passiert. Es können Ereignisse bezüglich Arbeitsmappen, Tabellenblättern oder Zellbereichen stattfinden. Diese können mit VBA-Code verbunden werden, so dass automatisch weitere Aktionen folgen.

Klassenmodul

Der VBA-Code von Ereignissen muss in den bereits vorhandenen Klassenmodulen eingetragen werden. Nach Erzeugen einer neuen Arbeitsmappe gibt es bereits die Klassenmodule *DieseArbeitsmappe* und *Tabelle1* bis *Tabelle3*. Ein Doppelklick auf eines der Klassenmodule im Projekt-Explorer führt dazu, dass das betreffende Codefenster angezeigt wird.

Ereignis auswählen

Oberhalb des Codefensters wählen Sie in der linken Liste das Objekt (`Workbook` oder `Worksheet`), in der rechten Liste anschließend das Ereignis aus. Es wird ein leerer Prozedurrahmen angezeigt, in dem die Ereignisprozedur geschrieben werden kann.

Noch umfangreicher sind die Möglichkeiten der Ereignisprogrammierung bei eigenen Dialogfeldern. Diese werden in Kapitel 10, »Dialogfelder«, beschrieben.

2.5.1 Arbeitsmappe wird geöffnet

Mappe öffnen

Mit folgender Prozedur wird beim Öffnen der Arbeitsmappe gleichzeitig eine zweite Arbeitsmappe geöffnet:

```
Private Sub Workbook_Open()
    MsgBox ActiveWorkbook.Name
    Workbooks.Open "C:\Temp\Mappe3.xlsm"
End Sub
```

Zur Erläuterung:

Workbook_Open()

- Es wird die Ereignisprozedur `Workbook_Open()` durchlaufen.
- Nach einer Kontrollausgabe wird eine zweite Arbeitsmappe geöffnet.

Testen Sie die Prozedur, indem Sie Excel vollständig schließen und anschließend die Arbeitsmappe, die die oben angegebene Prozedur enthält, öffnen.

2.5.2 Arbeitsmappe wird geschlossen

Mappe schließen

Mit folgender Prozedur im Klassenmodul *DieseArbeitsmappe* wird eine Arbeitsmappe vor dem Schließen ohne weitere Nachfrage automatisch gespeichert:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    ThisWorkbook.Save
End Sub
```

Zur Erläuterung:

Workbook_BeforeClose

- Es wird die Ereignisprozedur `Workbook_BeforeClose()` durchlaufen.
- Verhalten ohne diese Prozedur: Falls der Benutzer die Arbeitsmappe ändert und schließt, wird er gefragt, ob er die Arbeitsmappe speichern möchte. Diese Prozedur umgeht das, weil das Ereignis *BeforeClose* vor der Nachfrage auftritt.
- Innerhalb der Prozedur wird die Arbeitsmappe mit Hilfe der Methode `Save()` gespeichert.

Testen Sie die Prozedur, indem Sie eine Änderung vornehmen und die Arbeitsmappe schließen.

2.5.3 Tabellenblatt wird aktiviert

Blatt aktivieren

Mit folgender Prozedur in einem Klassenmodul wird nach dem Aktivieren des betreffenden Tabellenblatts (durch den Benutzer oder durch VBA-Code) eine Zelle ausgewählt:

```
Private Sub Worksheet_Activate()
    MsgBox ActiveSheet.Name
    Range("C5").Select
End Sub
```

Zur Erläuterung:

Worksheet_Activate()

- Es wird die Ereignisprozedur `Worksheet_Activate()` zu einem Tabellenblatt durchlaufen.
- Nach einer Kontrollausgabe wird eine Zelle ausgewählt.

Testen Sie die Prozedur, indem Sie zwischen den verschiedenen Tabellenblättern der Arbeitsmappe hin und her wechseln. Das Ereignis tritt nur ein, falls die Tabelle, in deren Codefenster dieser VBA-Code steht, nicht bereits das aktivierte Tabellenblatt ist.

2.5.4 Zellauswahl wechselt

Zelle wechselt

Mit folgender Prozedur in einem Klassenmodul wird nach der Auswahl eines Zellbereichs in der betreffenden Tabelle die Adresse dieses Zellbereichs ausgegeben:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    MsgBox Target.Address
End Sub
```

Zur Erläuterung:

SelectionChange()

- Es wird die Ereignisprozedur `Worksheet_SelectionChange()` zu einem Tabellenblatt durchlaufen.

Target

- Die Prozedur liefert den ausgewählten Zellbereich als `Range` über die Objektvariable `Target`. Weitere Einzelheiten zu Objektvariablen in Abschnitt 6.4, »Arbeiten mit Objektvariablen«.
- Für den ausgewählten Zellbereich wird die Eigenschaft `Address` ausgegeben. Dabei handelt es sich um die absolute Bezeichnung der Zelladresse.

Testen Sie die Prozedur, indem Sie verschiedene zusammenhängende bzw. nicht zusammenhängende Zellbereiche (mit Hilfe der Taste Strg) auswählen.

2.5.5 Doppelklick auf Zelle

Doppelklick

Mit folgender Prozedur in einem Klassenmodul wird bei einem Doppelklick auf eine Zelle die betreffende Zelle formatiert:

```
Private Sub Worksheet_BeforeDoubleClick _
        (ByVal Target As Range, Cancel As Boolean)
    Target.Interior.Color = vbGreen
    Target.Borders.LineStyle = xlContinuous
    Target.Borders.Weight = xlThick
    Cancel = True
End Sub
```

Zur Erläuterung:

BeforeDoubleClick

- Die Ereignisprozedur `Worksheet_BeforeDoubleClick()` zu einem Tabellenblatt wird durchlaufen. Mit `Target` wird die Zelle bezeichnet, auf der der Mauszeiger beim Doppelklick steht.
- Muster und Rahmen der Zelle werden eingestellt.

- Über die logische Variable `Cancel` können Sie bestimmen, ob das eigentliche Ereignis, hier der Doppelklick, stattfinden soll. Mit `Cancel = True` wird die weitere Verarbeitung abgebrochen, es findet nur die Formatierung statt. Mit `Cancel = False` oder einfach ganz ohne diese Anweisung wird weiter fortgesetzt, also der Cursor zur Bearbeitung in die Zelle gesetzt. Logische Variablen werden in Abschnitt 3.2.3, »Datentypen«, noch genauer erläutert. **Cancel**

2.5.6 Tabellenblatt wurde neu berechnet

Mit folgender Prozedur in einem Klassenmodul wird eine Spaltenbreite optimal eingestellt, falls im Tabellenblatt eine Neuberechnung an irgendeiner Stelle stattfand: **Berechnung**

```
Private Sub Worksheet_Calculate()
    Range("G:G").Columns.AutoFit
End Sub
```

Zur Erläuterung:

- Es wird die Ereignisprozedur `Worksheet_Calculate()` zu einem Tabellenblatt durchlaufen. Dieses Ereignis tritt ein, falls z. B. eine Zelle einen neuen Wert bekommen hat, die in der Berechnungsformel für den Wert einer anderen Zelle vorkommt. **Worksheet_ Calculate()**
- Beispiel: In der Zelle G3 steht die Formel `=G1+G2`. Sobald sich G1 oder G2 ändert, tritt das Ereignis *Calculate* auf.
- Die Spaltenbreite wird in diesem Falle optimal auf die neuen Werte eingestellt.