

# Workshop: VBA-Programmierung mit MS Excel

1 Zellen und Zellenbereiche programmieren.....	1
1.1 Auf Zellen und Zellenbereiche zugreifen .....	1
1.1.1 Zellen markieren und aktivieren .....	2
1.1.2 Zugriff auf die aktuelle Zelle .....	3
1.1.3 Zugriff auf einen ausgewählten Bereich .....	4
1.1.4 Alternative Schreibweisen .....	4
1.1.5 Zugriff auf benannte Zellen.....	5
1.1.6 Zeilen und Spalten.....	6
1.1.7 Zellenbereiche vereinigen .....	6
1.1.8 Relativer Zugriff auf Zellen und Zellenbereiche .....	7
1.1.9 Weitere Zugriffsmöglichkeiten .....	8
1.2 Inhalte in Zellen einsetzen und abfragen .....	8
1.2.1 Werte zuweisen .....	8
1.2.1 Formeln einsetzen .....	9
1.3 Zellen einfügen und löschen .....	11
1.3.1 Zellinhalte löschen .....	11
1.3.2 Zellen löschen.....	12
1.3.3 Zellen einfügen .....	12
1.3.4 Zellen formatieren.....	12
2 Übungen.....	19
3 Lösungen.....	21

## 1 Zellen und Zellenbereiche programmieren

Das Bearbeiten von Zellen und Zellenbereichen stellt den Kern von Excel dar. Aber es ist gewöhnungsbedürftig. Deshalb erfahren Sie in dieser Lektion, wie

- auf Zellen und Zellenbereich zugegriffen wird,
- dafür Eingaben erfolgen,
- Zellen eingefügt, gelöscht und formatiert werden,
- Schriften geändert werden,
- Zahlenformate eingestellt werden.

### 1.1 Auf Zellen und Zellenbereiche zugreifen

.Die wichtigsten Eigenschaften und Methoden des **Range**-Objekts sind in Tab. 1 enthalten:

Eigenschaft		Methode	
Cells	Zellen	Copy	Kopieren
Column, Columns	Spalten	Cut	Ausschneiden
Row, Rows	Zeilen	PasteSpecial	Einfügen
Border	Rahmen	Clear	Inhalt löschen
Interior	Innenbereich	Delete	Objekt löschen
Offset	Versetzung	Find	Suchen
Resize	Größenänderung	Insert	Einfügen
Font	Zeichensatz	Select	Markieren
ColumnWidth	Spaltenbreite	AutoFill	Füllen (automatisch)

Tab. 1: Wichtige Eigenschaften und Methoden des Range-Objekts

# Workshop: VBA-Programmierung mit MS Excel

Eine zentrale Rolle beim Zugriff auf Zellen und Zellenbereiche spielt die **Range**-Auflistung. Sie kann aus einer einzelnen Zelle, einer Spalte, einer Zeile, einer ausgewählten Menge von Zellen oder sogar einem dreidimensionalen Bereich bestehen.

Die Bearbeitung von Zellen oder Zellenbereichen kann entweder

- direkt erfolgen oder
- nach einer vorhergehender Selektion der Zellen.

Bei der direkten Bearbeitung folgt auf eine **Range**-Eigenschaft, die eine Auflistung zurückgibt, die gewünschte Eigenschaft. Beispiel:

```
Range("A1:B5").Font.Bold = True
```

Bei der zweiten Möglichkeit wird der Bereich durch die Methode **select** markiert. Der Zugriff auf diesen Bereich erfolgt dann mit der **selection**-Eigenschaft. Fortführung des Beispiels:

```
Range("A1:B5").Select  
Selection.Font.Bold = True
```

Diese Möglichkeit wird von Excel bei der Aufzeichnung von Mitschreibmakros bevorzugt. Der direkte Zugriff ist jedoch schneller.

## 1.1.1 Zellen markieren und aktivieren

Sind in einem Tabellenblatt Zellen markiert, gibt die **selection**-Eigenschaft das selektierte Objekt in der aktuellen Arbeitsmappe zurück. Die **activate**-Methode wird eigentlich nur benötigt, wenn innerhalb einer Markierung eine Zelle aktiviert werden soll. Fortführung des Beispiels:

```
Sub ZellenMarkieren()  
    With Worksheets("Tabelle1")  
        .Range("A1:B5").Select  
        .Range("B2").Activate  
    End With  
End Sub
```

Falls nur verwendete Zellen markiert werden sollen, wird mit der **UsedRange**-Eigenschaft gearbeitet. Beispiel: Die Tabelle `Tabelle1` der aktuellen Arbeitsmappe enthält folgende Angaben:

	A	B
1	Quartal	Umsatz
2	1	50.000,00 €
3	2	55.000,00 €
4	3	48.000,00 €
5	4	60.000,00 €

Abb. 1 Benutze Zellen markieren

Die folgende Prozedur markiert den gefüllten Bereich:

# Workshop: VBA-Programmierung mit MS Excel

```
Sub VerwendteZellenMarkieren()  
    With Worksheets("Tabelle1")  
        .Activate  
        .UsedRange.Select  
    End With  
End Sub
```

Angenommen, in Abb. 1. wäre die Zelle C1 aktiv. Ausgehend von C1 werden im folgenden Beispiel die angrenzenden Zellen markiert.

```
Sub AngrenzendeZellenMarkieren()  
    With Worksheets("Tabelle1")  
        .Activate  
        .Range("C1").Select  
    End With  
    ActiveCell.CurrentRegion.Select  
End Sub
```

## 1.1.2 Zugriff auf die aktuelle Zelle

Auf die aktive Zelle wird mit der **ActiveCell**-Eigenschaft zugegriffen. Sie gibt ein **Range**-Objekt zurück. Haben Sie beispielsweise auf Zelle A1 mit

```
Range("A1").Activate  
oder  
Range("A1").Select
```

zugegriffen, dann greifen Sie mit **ActiveCell** auf diese aktive Zelle zu, um ihr einen Wert zuzuordnen oder sie zu formatieren:

```
Sub ZugriffAufAktiveZelle()  
    Range("A1").Select  
    With ActiveCell  
        .Value = "Wert"  
        .Font.Bold = True  
    End With  
End Sub
```

Die aktive Zelle A1 wird mit folgender Prozedur eine Zeile nach unten verschoben, also nach A2:

```
Sub AktiveZelleEineZeileNachUnten()  
    Range("A1").Select  
    ActiveCell.Offset(1, 0).Select  
End Sub
```

Die Adresse der aktuellen Zelle sowie die aktuelle Zeilen- und Spaltennummer werden durch folgende Prozedur ins Direktfenster ausgegeben:

# Workshop: VBA-Programmierung mit MS Excel

```
Sub AktiveZelleEineZeileNachUnten()  
  Range("A1").Select  
  With ActiveCell  
    .Offset(1, 0).Select  
    Debug.Print "Adresse der aktuellen Zelle: " & .Address  
    Debug.Print "Aktive Zeile: " & .Row  
    Debug.Print "Aktive Spalte:" & .Column  
  End With  
End Sub
```

Grundsätzlich können alle Methoden und Eigenschaften des **Range**-Objekts auf die aktive Zelle **ActiveCell** angewandt werden.

## 1.1.3 Zugriff auf einen ausgewählten Bereich

Werden mehrere Zellen im aktiven Fenster markiert, so kann die Auswahl durch **Selection** spezifiziert werden. Mit der **select**-Methode kann beispielsweise der Zellenbereich A1 bis B5 ausgewählt werden. Um die markierten Zellen zu zählen, schreibt man:

```
Sub MarkierteZellenZaehlen()  
  With Worksheets("Tabelle1")  
    .Range("A1:B5").Select  
    MsgBox "Zahl der markierten Zellen: " & Selection.Count  
  End With  
End Sub
```

## 1.1.4 Alternative Schreibweisen

Die allgemein übliche Form der **Range**-Eigenschaft lautet folgendermaßen:

**Objekt.Range(Cell)**

Tab. 2 enthält typische Beispiele für die Argumente der **Range**-Eigenschaft in der üblichen Schreibweise:

Übliche Schreibweise	Bedeutung
Range("A1")	Einzelne Zelle A1
Range("A1:B4")	Zellen A1 bis B4
Range("A1, B4, C5")	Zellen A1, B4, C5
Range("C5:D9, G9:H16")	Mehrfachmarkierung eines Bereichs
Range("A:A")	Spalte A
Range("A:C")	Spalten A bis C
Range("1:1")	Zeile eins
Range("1:5")	Zeilen eins bis fünf
Range("1:1, 3:3, 8:8")	Zeilen eins, drei und acht
Range("A:A, C:C, F:F")	Spalten A, C und F

Tab. 2: Typische Argumente der **Range**-Eigenschaft und ihre Bedeutung

Eine zweite Möglichkeit, die **Range**-Eigenschaft zu schreiben, stellt sich wie folgt dar:

**Objekt.Range(Cell1, Cell2)**.

Diese alternative Form gibt immer einen rechteckigen Zellenbereich zurück. Abb. 3 enthält einen Vergleich beider Möglichkeiten. Trotz unterschiedlicher Schreibweise bewirken sie jeweils die Auswahl des gleichen Zellenbereichs:

# Workshop: VBA-Programmierung mit MS Excel

Übliche Schreibweise	Alternative Schreibweise
Range("A1:B4")	Range("A1", "B4")
Range("A1:D4")	Range("A1:B4", "D1:D4")

Tab. 3: Vergleich der Schreibweisen für die **Range**-Eigenschaft

Für die **Range**-Eigenschaft gibt es auch noch eine vereinfachte Schreibweise. Die **Range**-Eigenschaft kann mit eckigen Klammern einfacher geschrieben werden, wie folgendes Beispiel demonstriert:

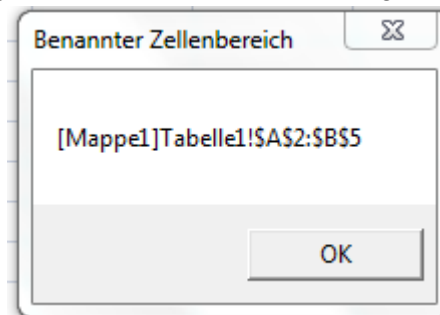
```
Sub VereinfachteRangeSchreibweise()  
    Dim intAnzahl As Integer  
    With Worksheets("Tabelle1")  
        .[A1:B5].Select  
        .[A6] = "Zahl der markierten Zellen: " & Selection.Count  
        intAnzahl = .[A1:A4, C1:C4].Count  
        .[A7] = "Anzahl: " & intAnzahl  
    End With  
End Sub
```

## 1.1.5 Zugriff auf benannte Zellen

Zellen und Zellenbereiche lassen sich mit Namen benennen. Solche Namen können auch innerhalb einer Prozedur festgelegt werden. Die folgende Prozedur `ZellenBereichBenennen` belegt die Zellen A2 bis B5 mit dem Namen *Quartalszahlen*.

```
Sub ZellenBereichBenennen()  
    With Worksheets("Tabelle1")  
        .Range("A2:B5").Name = "Quartalszahlen"  
    End With  
    MsgBox Range("Quartalszahlen").Address(External:=True), _  
        vbInformation, "Benannter Zellenbereich"  
End Sub
```

Das Ergebnis der Benennung wird in einer `MsgBox` wie folgt angezeigt:



Der Name *Quartalszahlen* wird wieder gelöscht mit:

```
Sub NameLoeschen()  
    With Worksheets("Tabelle1")  
        .Range("A2:B5").Name.Delete  
    End With  
End Sub
```

# Workshop: VBA-Programmierung mit MS Excel

## 1.1.6 Zeilen und Spalten

Um mit Spalten zu arbeiten, gibt es auch die Eigenschaften **Column**, **Columns** und **EntireColumn**. Für Zeilen existieren die vergleichbaren Eigenschaften **Row**, **Rows** und **EntireRow**.

Eigenschaft	Bedeutung
Column	Gibt eine Nummer zurück, die die erste Spalte in dem Bereich angibt, auf den die Eigenschaft bezogen wird.
Columns	Gibt ein Range-Objekt zurück, das die Spalten im angegebenen Bereich darstellt.
EntireColumn	Gibt ein Range-Objekt zurück, das eine oder mehrere ganze Spalten darstellt, die im angegebenen Bereich enthalten sind

Die Prozedur **ColumnProp** gibt in einer **MsgBox** die Zahl 2 zurück, weil es sich bei der Spalte B (die erste im markierten Bereich) um die zweite Spalte handelt.

```
Sub ColumnProp()  
    With Worksheets("Tabelle1")  
        .[B2:E4].Select  
    End With  
    MsgBox Selection.Column  
End Sub
```

Mit der Prozedur **ColumnsProp** wird die erste Spalte des ersten Tabellenblatts markiert:

```
Sub ColumnsProp()  
    With Worksheets("Tabelle1")  
        .Columns(1).Select  
    End With  
    MsgBox "Die Auswahl enthält " & _  
        Selection.Columns.Count & " Spalte(n)."  
End Sub
```

Die Meldung gibt an, wie viele Spalten (hier natürlich 1) die Markierung enthält.

**EntireColumn** gibt die gesamte(n) Spalte(n) des angegebenen Zellenbereichs zurück:

```
Sub MarkiereGanzeSpalten()  
    Worksheets("Tabelle1").Range("A1,D1:E1,G1").EntireColumn.Select  
End Sub
```

Die Prozedur **MarkiereGanzeSpalten** markiert die vier Spalten A, D, E und G.

## 1.1.7 Zellenbereiche vereinigen

Wie in Tab. 2 und Tab. 3 gezeigt, lassen sich mehrere Zellen oder Zellenbereiche zu einem Range-Objekt zusammenfassen. Das kann auch mit der **Union**-Methode bewirkt werden. Der einzige Unterschied besteht darin, dass **Range** als Argument eine Zeichenkette erwartet, während die **Union** als Argument **Range**-Objekte erwartet. Die folgende Prozedur **zellenvereinigen** verdeutlicht die Anwendung der **Union**-Methode:

# Workshop: VBA-Programmierung mit MS Excel

```
Sub ZellenVereinigen()  
    Dim rngR1 As Excel.Range  
    Dim rngR2 As Excel.Range  
    With Worksheets("Tabelle1")  
        Set rngR1 = .Range("A1:A6")  
        Set rngR2 = .Range("C1:C6")  
    End With  
    Union(rngR1, rngR2).Select  
    Set rngR1 = Nothing: Set rngR2 = Nothing  
End Sub
```

Die Anweisung **Union([A1:A6], [C1:C6]).Select** bewirkt das Gleiche wie obige Prozedur.

## 1.1.8 Relativer Zugriff auf Zellen und Zellenbereiche

Mit der **Offset**-Eigenschaft lässt sich der relative Zugriff auf Zellen ziemlich einfach durchführen. Sie lautet allgemein:

**Objekt.Offset(RowOffset, ColumnOffset)**

Sie gibt ein **Range**-Objekt zurück, das einen Bereich darstellt, der gegenüber dem angegebenen Bereich versetzt ist. Als Argumente können positive und negative Zahlen oder null verwandt werden. Die Prozedur **RelativerZugriff** verschiebt die aktuelle Zelle von A1 nach B3:

```
Sub RelativerZugriff()  
    Range("A1").Select  
    ActiveCell.Offset(RowOffset:=2, ColumnOffset:=1).Activate  
End Sub
```

Positive (negative) Werte für **RowOffset** bewirken eine Verschiebung unten (oben). Positive (negative) Werte für **ColumnOffset** bewirken eine Verschiebung nach rechts (links).

In einer Prozedur muss bei negativen Parametern in Form eines Fehlerausgangs abgefangen werden, dass der linke oder der oberen Tabellenrand nicht überschritten wird.

Mit **offset** kann nicht nur auf einzelne versetzte Zellen zugegriffen werden. Damit können aber auch ganze Zellenbereiche angesprochen werden, wie folgende Prozedur zeigt:

```
Sub BereichRelativAdressieren()  
    Dim rngBereich As Excel.Range  
    With Worksheets("Tabelle1")  
        Set rngBereich = .Range("A1:A5")  
    End With  
    rngBereich.Offset(ColumnOffset:=2) = "Hallo"  
    Set rngBereich = Nothing  
End Sub
```

Diese Prozedur setzt in jeder der Zellen C1 bis C5 das Wort *Hallo* ein.

# Workshop: VBA-Programmierung mit MS Excel

## 1.1.9 Weitere Zugriffsmöglichkeiten

Auch mit Hilfe der **Cells**-Eigenschaft kann ein Range-Objekt zurückgegeben werden. Der Vorteil der **Cells**-Eigenschaft gegenüber der **Range**-Eigenschaft ergibt sich aus der Syntax:

```
Objekt.Cells (RowIndex, ColumnIndex)
```

Für die **Cells**-Eigenschaft können numerische Werte angegeben werden. Mit der folgenden Prozedur **ZellenAdressieren** wird zum Beispiel der Schriftgrad der Zelle C5 in **Tabelle1** auf 14 Punkte festgelegt.

```
Sub ZellenAdressieren()  
    Worksheets("Tabelle1").Cells(5, 3).Font.Size = 14  
End Sub
```

Es gibt noch zwei weitere Formen, die mit der **Cells**-Eigenschaft angewandt werden können:

- **Objekt.Cells (Index)**
- **Objekt.Cells**

Bei der zuerst genannten Form werden die Zeilen *zeilenweise* gezählt. Mit dem Parameter **Index** wird die Zielzelle angegeben. Dazu zwei Beispiele:

<code>Worksheets(1).Cells(3)</code>	Zelle C1 wird adressiert
<code>Worksheets(1).Cells(259)</code>	Zelle C2 wird adressiert ( $256 + 3 = 259$ )

Die zuletzt benannte Form beschreibt alle Zellen eines Tabellenblatts. Mit der folgenden Prozedur wird der Inhalt aller Zellen des ersten Tabellenblatts horizontal ausgerichtet:

```
Sub AllesHorizontalZentrieren()  
    Worksheets(1).Cells.HorizontalAlignment = xlCenter  
End Sub
```

Die **Range**- und die **Cells**-Eigenschaft können zusammen eingesetzt werden, wie aus folgender Prozedur entnommen werden kann:

```
Sub SchriftfarbeBlau()  
    Const conBlue As Long = 5  
    Worksheets(1).Range(Cells(1, 1), Cells(3, 3)).Font.ColorIndex = conBlue  
End Sub
```

Für den Zellenbereich A1 bis C3 wird damit die Schriftfarbe blau festgelegt.

## 1.2 Inhalte in Zellen einsetzen und abfragen

### 1.2.1 Werte zuweisen

Zur Zuweisung eines Wertes oder eines Textes an eine Zelle sowie zum Abfragen ihres Inhalts wird die Eigenschaft **Value** benutzt. In der Prozedur **WerteZuwweisen** werden den Zellen A1 bis A4 Werte zugewiesen und anschließend wieder ausgelesen.



# Workshop: VBA-Programmierung mit MS Excel

```
Sub WerteZuweisen()  
    Dim strMsg As String  
    Dim intZahl As Integer  
    Dim intLoop As Integer  
    With Worksheets(1)  
        ' Inhalte einsetzen  
        .Range("A1").Value = "Zahlen"  
        ' vereinfachte Schreibweise  
        .[A2].Value = 12  
        .[A3].Value = 24  
        .[A4].Value = 36  
        ' Inhalte entnehmen  
        strMsg = .[A1].Value & ": "  
        ' Zählschleife über drei Zellen  
        For intLoop = 1 To 3  
            ' Inhalt zwischenspeichern  
            intZahl = .Range("A" & intLoop + 1).Value  
            strMsg = strMsg & intZahl & ", "  
        Next intLoop  
    End With  
    strMsg = Left(strMsg, Len(strMsg) - 2)  
    ' Ergebnis anzeigen  
    MsgBox strMsg, vbInformation, "Zellinhalte setzen und abfragen"  
End Sub
```

Bei der **Value**-Eigenschaft handelt es sich um eine Standardeigenschaft. Sie ist deshalb optional und kann weggelassen werden. Wird keine Eigenschaft angegeben, wird automatisch **Value** verwendet.

## 1.2.1 Formeln einsetzen

Eine Formel wird in eine Zelle eingesetzt, indem sie einfach als Zeichenkette übergeben wird. Beispiel:

```
Sub FormelnEinsetzen()  
    With Worksheets(1)  
        ' Addieren  
        .[B1] = 10  
        .[B2] = 5  
        .[B3].Value = "=B1+B2"  
        ' Dividieren  
        .[C1] = 96  
        .[C2] = 12  
        .[C3].Formula = "=C1/C2"  
        ' Summieren  
        .[D1] = 96  
        .[D2] = 12  
        .[D3] = 13  
        .[D4] = "=Round(Sum(D1:D3),2)"  
        .[D5].FormulaLocal = "=Runden(Summe(D1:D5);2)"  
        ' arithm. Mittel  
        .[F4] = "=Round(Average(D1:D3),2)"  
        .[F5].FormulaLocal = "=Runden(Mittelwert(D1:D3);2)"  
        ' Tagesdatum setzen  
        .[G1] = "=Now()"  
    End With  
End Sub
```

# Workshop: VBA-Programmierung mit MS Excel

```
. [G2].FormulaLocal = "=Jetzt()"
' Bestimmtes Datum setzen
. [G3].FormulaLocal = "=(Datwert ("22.2.2010"))"
' Englischen Funktionsnamen anzeigen
. [G3].Select
  MsgBox "Englischer Name der Funktion von 'Datwert': " _
    & ActiveCell.Formula
End With
End Sub
```

Wenn die Eigenschaft **FormulaLocal** benutzt wird, muss der Name der Tabellenfunktion deutsch übergeben werden. Mit **ActiveCell.Formula** kann der dazugehörige englische Funktionsname ermittelt werden, siehe Abb. 2.

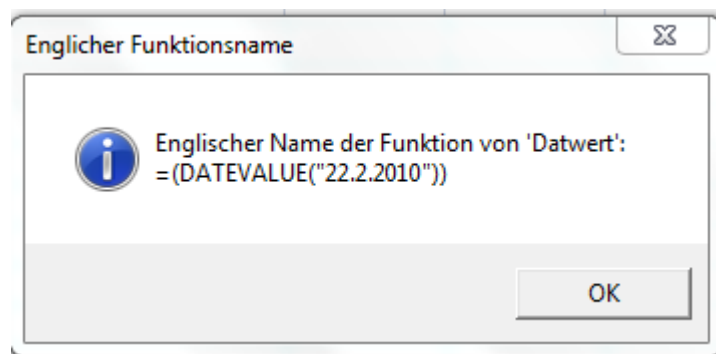


Abb. 2: Englischen Funktionsnamen von **Datwert** anzeigen

Manchmal eignet sich die sogen. Z1S1 bzw. R1C1-Schreibweise besser zum Eingeben einer Formel. Dabei steht

- Z für Zeile,
- S für Spalte,
- R für Row,
- C für Column

Diese Schreibweise fügt die Zellbezüge relativ zu der Zelle ein, die in der Formel enthalten ist. Beispiel:

```
Sub RelativerBezug()
  Dim strMsg As String
  With Worksheets(1)
    ' englisch
    .[A1] = 10
    .[A2] = 5
    .[A3].Formula = "=Sum(A1+A2)"
    .[A3].Select
    strMsg = .[A3].Formula & vbNewLine & _
      .[A3].FormulaR1C1 & String(2, vbNewLine)
    ' deutsch
    .[B1] = 10
    .[B2] = 5
    .[B3].FormulaLocal = "=Summe(B1+B2)"
    .[B3].Select
    strMsg = strMsg & .[B3].FormulaLocal & vbNewLine & _
      .[B3].FormulaR1C1Local
  End With
  MsgBox strMsg, vbInformation, "Absolute u. relative Zellbezüge"
```

# Workshop: VBA-Programmierung mit MS Excel

```
End With  
End Sub
```

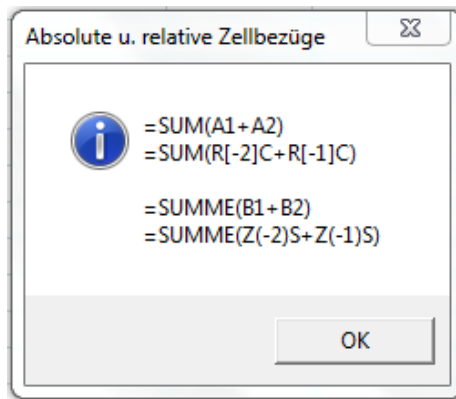


Abb. 3: Meldung der Prozedur RelativerBezug

Wie Datumwerte an Zellen übergeben werden, zeigt folgende Prozedur:

```
Sub DatumswerteEingeben()  
    Dim strMsg As String  
    With Worksheets(1)  
        ' Datum eingeben  
        .[A1] = #2/17/2011#  
        .[A2] = CDate("17.2.2011")  
        ' Tagesdatum  
        .[A3] = Date ' Tagesdatum  
        ' Systemzeit  
        .[A4] = Time  
        .[A4].NumberFormat = "h:mm"  
        If IsEmpty([A5]) Then  
            MsgBox "Zelle A5 ist leer"  
        End If  
    End With  
End Sub
```

Abb. 4 beinhaltet die Ausgaben der obigen Prozedur.

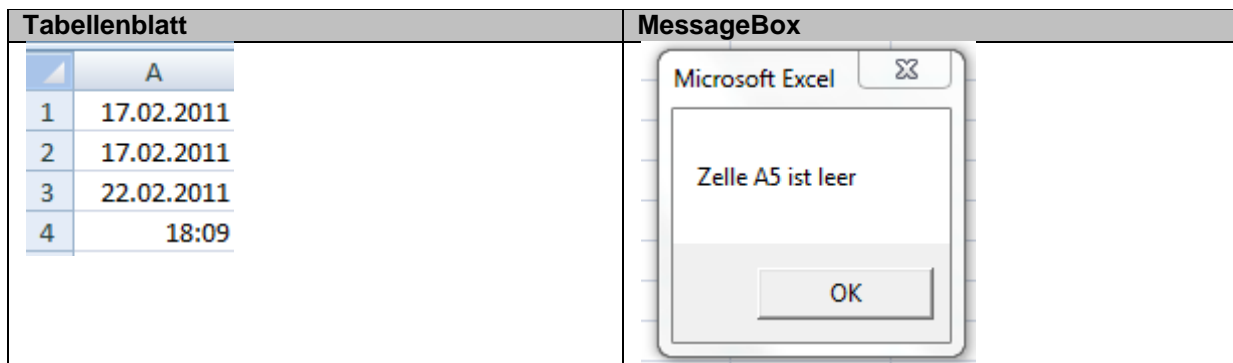


Abb. 4: Ausgaben der Prozedur DatumswerteEingeben

## 1.3 Zellen einfügen und löschen

### 1.3.1 Zellinhalte löschen

- **ClearContents** löscht den Inhalt einer Zelle.

# Workshop: VBA-Programmierung mit MS Excel

- **Clear** löscht sowohl den Inhalt als auch die Formatierung.
- **ClearFormats** löscht die Formatierung.
- **ClearNotes** löscht Notizen zu Zellen

## 1.3.2 Zellen löschen

Mit **Delete** werden Zellen gelöscht. Der allgemeine Ausdruck lautet:

**Ausdruck.Delete(Shift)**

Dabei muss **Ausdruck** eine Eigenschaft sein, die ein **Range**-Objekt zurückgibt. Der Parameter **Shift** ist optional. Code-Beispiel:

```
With Worksheets(1)
    ' Zellen von rechts nachrücken lassen
    .[A1:A3].Delete Shift:=xlShiftToLeft
    ' sonst Zellen nach oben rücken
    .[A1:C3].Delete Shift:=xlShiftUp
End
```

Soll eine ganze Zeile oder Spalte gelöscht werden, dann codieren Sie beispielsweise so:

```
With Worksheets(1)
    .[A1].Select
    ActiveCell.EntireColumn.Delete
    .[B1].EntireRow.Delete
End With
```

## 1.3.3 Zellen einfügen

Neue Zellen werden mit der **Insert**-Methode in ein Tabellenblatt eingefügt. Auch dabei gibt es das Argument **Shift** mit den beiden Konstanten **xlDown** und **xlToRight**. **Insert** wird auf ein **Range**-Objekt angewandt.

```
With Worksheets(1)
    .[A1].Select
    ActiveCell.EntireColumn.Insert
    .[A1:C3].Insert Shift:=xlToRight
    .Rows(3).Insert
End With
```

## 1.3.4 Zellen formatieren

In diesem Abschnitt geht es darum, Zellen zu formatieren, d. h.

- Hintergrundfarbe setzen
- Muster und Rahmen verwenden
- Ausrichtung einer Zelle festlegen
- Schriftart und Schriftattribute bestimmen
- Benutzerdefinierte Formate einsetzen.

# Workshop: VBA-Programmierung mit MS Excel

## 1.3.4.1 Hintergrundfarbe setzen

Die Prozedur **FarbenAusgeben** zeigt, wie die Zellen eines Tabellenblatts eingefärbt werden können. Die Prozedur gibt 56 Hintergrundfarben mit dem jeweils dazugehörigen Farbindex (**ColorIndex**) aus (siehe Abb. 3).

```
Sub FarbenAusgeben()  
    Dim intFarbe As Integer  
    Dim intSpalte As Integer  
    Dim intZeile As Integer  
    Dim rng As Range  
    With Worksheets(1)  
        ' Spaltenbreite setzen  
        .Range("B:B, D:D, F:F, H:H").ColumnWidth = 7.5  
        .Range("A:A, C:C, E:E, G:G").ColumnWidth = 3  
        ' Farbe entfernen  
        .[A1:H14].Interior.ColorIndex = xlNone  
        .[A1].Select  
        Set rng = ActiveCell  
        ' Zellen einfärben  
        For intFarbe = 0 To 55  
            intSpalte = (intFarbe \ 14) * 2 ' ganzzahlige Division  
            intZeile = intFarbe Mod 14 ' Divisionsrest best.  
            With rng  
                .Offset(intZeile, intSpalte) = intFarbe + 1  
                .Offset(intZeile, intSpalte + 1) _  
                    .Interior.ColorIndex = intFarbe + 1  
            End With  
        Next intFarbe  
    End With  
End Sub
```

Die Ergebnis der Prozedur **FarbenAusgeben** ist in Abb. 3 zu sehen.

	A	B	C	D	E	F	G	H
1	1		15		29		43	
2	2		16		30		44	
3	3		17		31		45	
4	4		18		32		46	
5	5		19		33		47	
6	6		20		34		48	
7	7		21		35		49	
8	8		22		36		50	
9	9		23		37		51	
10	10		24		38		52	
11	11		25		39		53	
12	12		26		40		54	
13	13		27		41		55	
14	14		28		42		56	

Abb. 3: Tabellenblatt mit 56 Hintergrundfarben

# Workshop: VBA-Programmierung mit MS Excel

## 1.3.4.2 Muster und Rahmen verwenden

Muster lassen sich über einen Index aus der Excel-Musterpalette auswählen. Neben dem Muster muss die Hintergrundfarbe sowie die Farbe des Musters festgelegt werden. Das folgende Codebeispiel belegt die Zellen B1 bis B5 mit gelben Streifen auf blauem Grund.

```
With Worksheets(1)
  With .Range("B1:B5").Interior
    .Pattern = 6          ' Muster (
    .PatternColorIndex = 6 ' Farbe des Musters (gelb)
    .ColorIndex = 5      ' Hintergrundfarbe (blau)
  End With
End With
```

Das zugehörige Ergebnis ist in Abb. 4 zu sehen.

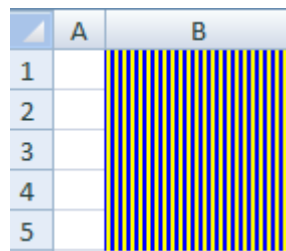


Abb. 4: Muster für ausgewählte Zellenbereiche

Um beispielsweise eine mittelstarke, gestrichelte Linie in rot um den Zellenbereich B2:D4 zu ziehen, wird geschrieben:

```
Worksheets(1).Range("B2:D4").BorderAround _
  LineStyle:=xlDash, ColorIndex:=3, Weight:=xlMedium
```

Das dazugehörige Ergebnis zeigt Abb. 5.

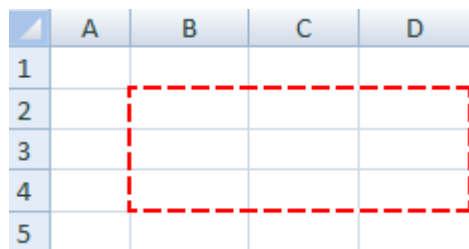


Abb. 5: Rot eingerahmter Zellenbereich

Soll nicht der angegebene Bereich an sich, sondern die einzelnen Zellen in diesem Bereich umrahmt werden, wird die **Borders**-Eigenschaft verwendet. Das folgende Code-Beispiel besagt, dass alle Zellen im Bereich B2 bis D4 mit einer gestrichelten, mittelstarken, roten Linie umrahmt werden sollen.

```
With Worksheets(1).Range("B2:D4").Borders
  .LineStyle = xlDash ' gestrichelt
  .ColorIndex = 3     ' rot
  .Weight = xlMedium ' mittelstarker Strich
End With
```

Das Ergebnis der Prozedur wird in Abb. 6 gezeigt:

# Workshop: VBA-Programmierung mit MS Excel

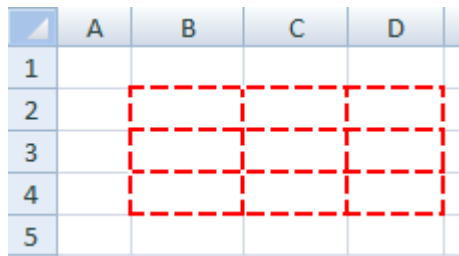


Abb. 6: Rot eingerahmte Zellen

Die **Borders**-Eigenschaft kann auch mit einem Index eingesetzt werden, der einen bestimmten Teil des Rahmens beschreibt, beispielsweise **Borders (xlEdgeTop)** für die obere Begrenzungslinie des Rahmens.

### 1.3.4.3 Ausrichtung einer Zelle festlegen

Für Zellen lässt sich eine horizontale und vertikale Ausrichtung des jeweiligen Inhalts festlegen. Code-Beispiel:

```
With Worksheets(1).Range("A1:D4")  
    .VerticalAlignment = xlVAlignCenter  
    .HorizontalAlignment = xlHAlignCenter  
End With
```

Damit wird der Zellenbereich von A1 bis D4 horizontal und vertikal mittig ausgerichtet.

Die Eigenschaft **WrapText** erlaubt das Ein- und Ausschalten eines Zeilenumbruchs. Die Eigenschaft **Orientation** ermöglicht das Drehen des Inhalt von Zellen. Dazu ein Code-Beispiel:

```
With Worksheets(1)  
    With .[A1:B1]  
        .WrapText = True ' Zeilenumbruch  
        .Orientation = 45 ' Drehung in Grad  
    End With  
    .[A1].Value = "Drehung"  
    .[B1].Value = "45 Grad"  
End With
```

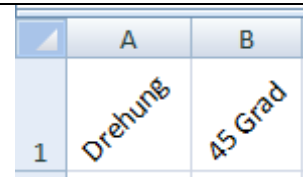


Abb. 7: Text in Zellen drehen

### 1.3.4.4 Schriftart und Schriftattribute bestimmen

Die Veränderung von Schriftart und Schriftattributen erfolgt mit Eigenschaften des **Font**-Objekts. Code-Beispiel: Einem Zellenbereich [A5:E5] namens **summen** soll die Schriftart Arial in der Größe 20 Punkte in *kursiver* Fettschrift zugewiesen werden. Um manuelle Eingaben zu ersparen, sollen im genannten Zellenbereich ganzzahlige Zufallszahlen zwischen 100 und 1000 als fiktive Summen eingesetzt werden. Abb. 8 illustriert das Ergebnis.

```
With Worksheets(1)  
    Const conUntergrenze As Integer = 100  
    Const conObergrenze As Integer = 1000  
    Dim intZaehler As Integer  
    ' Ganzzahlige Zufallszahlen erzeugen zwischen 100 und 1000  
    For intCount = 1 To 5  
        .Cells(5, intCount) = _  
            Int((conObergrenze - conUntergrenze + 1) * Rnd + conUntergrenze)
```

# Workshop: VBA-Programmierung mit MS Excel

```
Next intCount
With .[A5:E5]      ' Bereich
    .Name = "Summen" ' Bereichsname
    .ColumnWidth = 9.5 ' Spaltenbreite
    With [Summen].Font
        .Name = "Arial" ' Schriftart
        .Size = 20 ' Schriftgröße
        .Bold = True ' Fettdruck
        .Italic = True ' Kursivdruck
    End With
End With
End With
End With
```

	A	B	C	D	E
1					
2					
3					
4					
5	<b>145</b>	<b>514</b>	<b>963</b>	<b>891</b>	<b>473</b>

Abb. 8: Schriftart und Schriftattribute bestimmen

## 1.3.4.5 Benutzerdefinierte Formate einsetzen

Das folgende Code-Beispiel demonstriert, wie Zahlenwerte formatiert werden können. Unter den Titel *Benutzerdefinierte numerische Formate* können in der Excel-Hilfe die Formatierungszeichen für Zahlenwerte nachgeschlagen werden. In der folgenden Prozedur **Zahlenformate** sind acht benutzerdefinierte Zahlenformate gelb markiert.

```
Sub Zahlenformate()
With Worksheets(1)
    .Cells.Clear
    .[A1].Select
    With ActiveCell
        .Value = 3.142
        .NumberFormatLocal = "0,000"
        .Offset(ColumnOffset:=1).Value = "Zahl mit drei Nachkommastellen"
    End With
    .[A2].Select
    With ActiveCell
        .Value = 1234
        .NumberFormatLocal = "€ 0"
        .Offset(ColumnOffset:=1).Value = "Zahl mit führendem Euro-Zeichen"
    End With
    .[A3].Select
    With ActiveCell
        .Value = 1234
        .NumberFormatLocal = "€ * 0"
        .Offset(ColumnOffset:=1).Value = "Euro-Zeichen links, Zahl rechts in der Zelle"
    End With
    .[A4].Select
    With ActiveCell
        .Value = 12345.99
        .NumberFormatLocal = "€ * #.##0,00"
        .Offset(ColumnOffset:=1).Value = "Euro-Zeichen links, Zahl mit Tausenderzeichen und zwei Dezimalstellen"
    End With
    .[A5].Select
    With ActiveCell
        .Value = 13.52
```



# Workshop: VBA-Programmierung mit MS Excel

```
.NumberFormatLocal = "[blau]###0,00 "€"/kg""
.Offset(ColumnOffset:=1).Value = "blaue Zahl mit Tausenderzeichen, zwei Dezimalstellen
und Text €/kg"
End With
With [A6:A9]
.NumberFormatLocal = "0,0???"
.Offset(ColumnOffset:=1).Value = "Zahlen am Dezimalkomma ausrichten"
End With
.[A6].Select
With ActiveCell
.Value = 0.3
.Offset(RowOffset:=1).Value = 0.3333
.Offset(RowOffset:=2).Value = 0.535
.Offset(RowOffset:=3).Value = 0.09
End With
.[A10].Select
With ActiveCell
.NumberFormatLocal = "@"
.Value = "12345,678"
.HorizontalAlignment = xlRight
.Offset(ColumnOffset:=1).Value = "Text als Zahl darstellen"
End With
With [A11:A13]
.NumberFormatLocal = "#.##0,00 €;-#.##0,00 €; ""*"" €"
.Offset(ColumnOffset:=1).Value = "unterschiedliche Formate für positive u. negative Zahlen"
End With
.[A11] = 987.65
.[A12] = -987.65
.[A13] = 0
With [A14:A16]
.NumberFormatLocal = "[<>]0 ""Tage""; 0 ""Tag""
.Offset(ColumnOffset:=1).Value = "Bedingtes Zahlenformat"
End With
.[A14] = 0
.[A15] = 1
.[A16] = 2
.Columns(1).AutoFit ' Optimale Spaltenbreite
.Columns(2).AutoFit
With .Range("A6:B9") ' Bereich einrahmen
.Borders(xlEdgeTop).LineStyle = xlContinuous
.Borders(xlEdgeBottom).LineStyle = xlContinuous
.Borders(xlEdgeRight).LineStyle = xlContinuous
End With
With .Range("A11:B13") ' Bereich einrahmen
.Borders(xlEdgeTop).LineStyle = xlContinuous
.Borders(xlEdgeBottom).LineStyle = xlContinuous
.Borders(xlEdgeRight).LineStyle = xlContinuous
End With
With .Range("A14:B16") ' Bereich einrahmen
.Borders(xlEdgeBottom).LineStyle = xlContinuous
.Borders(xlEdgeRight).LineStyle = xlContinuous
End With
End With
End Sub
```

Das Ergebnis der Prozedur veranschaulicht Abb. 9.

## Workshop: VBA-Programmierung mit MS Excel

	A	B
1	3,142	Zahl mit drei Nachkommastellen
2	€ 1234	Zahl mit führendem Euro-Zeichen
3	€ 1234	Euro-Zeichen links, Zahl rechts in der Zelle
4	€ 12.345,99	Euro-Zeichen links, Zahl mit Tausenderzeichen und zwei Dezimalstellen
5	13,52 €/kg	blaue Zahl mit Tausenderzeichen, zwei Dezimalstellen und Text €/kg
6	0,3	Zahlen am Dezimalkomma ausrichten
7	0,3333	Zahlen am Dezimalkomma ausrichten
8	0,535	Zahlen am Dezimalkomma ausrichten
9	0,09	Zahlen am Dezimalkomma ausrichten
10	12345,678	Text als Zahl darstellen
11	987,65 €	unterschiedliche Formate für positive u. negative Zahlen
12	-987,65 €	unterschiedliche Formate für positive u. negative Zahlen
13	* €	unterschiedliche Formate für positive u. negative Zahlen
14	0 Tage	Bedingtes Zahlenformat
15	1 Tag	Bedingtes Zahlenformat
16	2 Tage	Bedingtes Zahlenformat

Abb. 9: Ausgewählte benutzerdefinierte Zahlenformate

Um Datumswerte und Uhrzeiten darzustellen, stellt Excel eine Fülle spezieller Formatzeichen bereit. Dazu findet man unter dem Titel *Benutzerdefinierte Datums- und Zeitformate* in der Excel-Hilfe detaillierte Informationen. Im Folgenden wird lediglich das sogen. lange Datumsformat behandelt. Die unten stehende Prozedur `LangesDatumsFormat` gibt acht Datumswerte im Zellenbereich von A1 bis A8 in langer Schreibweise aus:

```
Sub LangesDatumsFormat()  
  With Worksheets(2)  
    Const conText As String = "Excel-Workshop"  
    Dim intDatum As Integer  
    Dim rng As Range  
    Dim lngLetzte As Long  
    .Cells.Clear  
    Set rng = .Range("A1")  
    ' Zellen in Spalte A mit Datumswerten belegen  
    With rng  
      .Value = #2/24/2011#  
      .Offset(ColumnOffset:=1) = conText  
      .EntireColumn.AutoFit  
    End With  
    intDatum = 1  
    Do  
      rng.Offset(RowOffset:=intDatum) = DateAdd("d", intDatum, rng.Value)  
      intDatum = intDatum + 1  
    Loop Until intDatum > 7  
    With .[A:A]  
      .NumberFormatLocal = "TTTT", den "T.M.JJJJ"  
      .EntireColumn.AutoFit  
    End With  
    With .[B:B]  
      .EntireColumn.AutoFit  
      With .Font  
        .Bold = True  
        .ColorIndex = 3  
      End With  
    End With  
  End With  
End Sub
```

# Workshop: VBA-Programmierung mit MS Excel

```
End With
End With
' Letzte belegte Zelle in Spalte A bestimmen
lngLetzte = .Cells(Rows.Count, 1).End(xlUp).Row
.Cells(lngLetzte, 2) = conText
Set rng = Nothing
End With
End Sub
```

Abb. 10 konkretisiert das Ergebnis der Prozedur.

	A	B
1	Donnerstag, den 24.2.2011	Excel-Workshop
2	Freitag, den 25.2.2011	
3	Samstag, den 26.2.2011	
4	Sonntag, den 27.2.2011	
5	Montag, den 28.2.2011	
6	Dienstag, den 1.3.2011	
7	Mittwoch, den 2.3.2011	
8	Donnerstag, den 3.3.2011	Excel-Workshop

Abb. 10: Langes Datumsformat in Spalte A

## 2 Übungen

1

Eine Prozedur namens **FormatierteZeileFuellen** soll die zweite Zeile des Tabellenblatts `Tabelle1` wie folgt formatieren und danach füllen:

- Zeilenhöhe: 20 Punkte
- Hintergrundfarbe: hellgelb (Farbindex: 36)
- Rahmenfarbe der einzelnen Zellen: blau
- Farbe der Umrandung: lavendel (Farbindex 39)
- Linienstärke der Umrandung: fett

Die erste Zelle der zweite Zeile soll die Zeichenkette *Montag* enthalten. Unter Verwendung der Methode **AutoFill** soll der Bereich von B2 bis G2 automatisch gefüllt werden.

Das Ergebnis der Prozedur soll wie folgt aussehen:

	A	B	C	D	E	F	G
1							
2	Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag	Sonntag
3							

2


Eine Prozedur namens **Summenformel** soll die Zellen A1 bis A10 mit den Zahlen 1 bis 10 füllen. In die Zelle A11 soll eine Formel zur Summierung der Zahlen in den genannten Zellen eingetragen werden. In einem Meldungsfenster soll schließlich die Formel und das Ergebnis

# Workshop: VBA-Programmierung mit MS Excel

der Summierung angezeigt werden. Die folgende Darstellung veranschaulicht die Aufgabenstellung.

	A	B	C	D	E	F
1	1					
2	2					
3	3					
4	4					
5	5					
6	6					
7	7					
8	8					
9	9					
10	10					
11	55					

Summenformel

 Summenformel: =SUMME(\$A\$1:\$A\$10)  
Ergebnis: 55

OK

# Workshop: VBA-Programmierung mit MS Excel

## 3 Lösungen

1

```
Sub FormatierteZeileFuellen()  
    Dim rngQuelle As Range ' Quellbereich  
    Dim rngZiel As Range ' Zielbereich  
    Worksheets("Tabelle1").Activate  
    Set rngZiel = [A2:G2]  
    ' Zielbereich formatieren  
    With rngZiel  
        .RowHeight = 20 ' Zeilenhöhe  
        .Interior.ColorIndex = 36 ' hellgelb  
        .Borders.Color = vbBlue ' blau  
        ' Umrandung: lavendel, fett  
        .BorderAround ColorIndex:=31, Weight:=xlThick  
    End With  
    Set rngQuelle = [A2]  
    With rngQuelle  
        .Value = "Montag"  
        .AutoFill Destination:=rngZiel, Type:=xlFillValues  
    End With  
    Set rngQuelle = Nothing: Set rngZiel = Nothing  
End Sub
```

2

```
Sub Summenformel()  
    Dim rngZellen As Range  
    Dim intZelle As Integer  
    Dim strMsg As String  
    Worksheets("Tabelle1").Activate  
    Set rngZellen = Range(Cells(1, 1), Cells(10, 1))  
    For intZelle = 1 To 10  
        rngZellen(intZelle, 1).Value = intZelle  
    Next intZelle  
    With Cells(11, 1)  
        .FormulaLocal = "=Summe(" & rngZellen.Address & ")"  
        .Borders(xlEdgeLeft).LineStyle = xlContinuous  
        .Borders(xlEdgeRight).LineStyle = xlContinuous  
        .Borders(xlEdgeTop).LineStyle = xlContinuous  
        .Borders(xlEdgeBottom).LineStyle = xlDouble  
        strMsg = "Summenformel: " & .FormulaLocal & vbNewLine & _  
            "Ergebnis: " & .Value  
    End With  
    MsgBox strMsg, vbInformation, "Summenformel"  
    Set rngZellen = Nothing  
End Sub
```

# Workshop: VBA-Programmierung mit MS Excel

```
Sub ZeileEinfügen()  
    Rows(ActiveCell.Row).Insert Shift:=xlDown  
End Sub
```

```
Sub ZeileLoeschen()  
    Rows(ActiveCell.Row).Delete Shift:=xlUp  
End Sub
```