Workshop 7: VBA-Programmierung mit MS Excel

1 Mit Tabellenblättern arbeiten

Arbeitsmappen (`Workbooks`) enthalten Blätter (**`Sheets`**), die entweder Arbeitsblätter (`Worksheet`) oder Diagrammblätter (`Chart`) darstellen.

1.1 Auf Blätter zugreifen

- Die `Sheets`-Auflistung enthält alle Arbeits –und Diagrammblätter einer Arbeitsmappe. Nur über diese Auflistung können neue Blätter angelegt werden, die dann in die Auflistung eingefügt werden. Damit können Aktionen durchgeführt werden, die sich allgemein auf die Verwaltung von Arbeits- und Diagrammblättern beziehen.
- `ActiveSheet` verweist auf das aktuelle Arbeitsblatt unabhängig vom Typ des Blatts. Achtung: Ein *ActiveWorksheet* gibt es nicht!
- `ActiveChart` ermöglicht den Zugriff auf das aktive Diagrammblatt.

- Über ein Element der `Sheets`-Auflistung wird auf ein konkretes Blatt verwiesen. Dabei erfolgt der Zugriff über einen Index oder über den Namen des Blattes: Beispiel:
  - `Sheets(2)`
  - `Sheets("Name")`

  Der Index entspricht der Reihenfolge der Blätter.
  Der Zugriff auf ein konkretes Blatt kann auch mit einer Objektvariablen vom Typ `Worksheet` oder `Chart` erfolgen. Beispiel:
  ```
  Dim objSht As Worksheet
  Dim objCht as Chart
  ```
- Wichtige Eigenschaften und Methoden der Sheets-Auflistung:
  - `Sheets.Count`: Diese Eigenschaft gibt einen `Long`-Wert zurück, der die Anzahl der Objekte in der Auflistung darstellt.
  - `Sheets.Add`: Diese Methode erstellt ein neues Arbeitsblatt, Diagramm oder Makroblatt. Das neue Arbeitsblatt wird zum aktiven Blatt.
  - `Sheets(2).Select`: Diese Methode markiert das angegebene Objekt.
  - `Sheets.Item(2)`: Diese Eigenschaft gibt das angegebene Blatt der Auflistung zurück
- Das folgende Beispiel veranschaulicht das Arbeiten mit einer Objektvariablen vom Typ `WorkSheet`:

```
Sub AufEinBlattVerweisen()
    Dim objSht As Worksheet ' Objektvariable deklarieren
    ' Objektvariable setzen
    Set objSht = Workbooks.Item("Mappe1").Sheets.Item("Tabelle1")
    With objSht
      ' Objekt benennen
      .Name = "Termine"
    End With
End Sub
```

1.2 Gemeinsame Eigenschaften des `Worksheet` und `Chart`-Objekts

Eigenschaft	Bedeutung
`Name`	Der Name eines Blattes kann ermittelt oder bestimmt werden. `Activesheet.Name`
`Next`	Nächsten Blatt aktivieren (vorwärts springen) `Activesheet.Next.Select`
`Previous`	Vorheriges Blatt aktivieren (rückwärts springen) `ActiveSheet.Previous.Select`

1.3 Spezielle Eigenschaften des `Worksheet`-Objekts

Eigenschaft	Bedeutung
`Cells`	Der Name eines Blattes kann ermittelt oder bestimmt werden. Beispiel: `MsgBox ActiveSheet.Cells(1, 1).Value`
`Columns`	Gibt eine `Range`-Objekt zurück, dass alle Spalten eines Arbeitsblatts enthält. Beispiel: `ActiveSheet.UsedRange.Columns.Count`
`Rows`	Gibt eine `Range`-Objekt zurück, dass alle Zeilen eines Arbeitsblatts enthält. Beispiel: `ActiveSheet.UsedRange.Rows.Count`

`FilterMode`	Ruft einen Wert ab, der angibt, ob sich das Arbeitsblatt im Filtermodus befindet. Beispiel:
	`Function IstFilter(objSht As Worksheet) As Boolean`
	`  IstFilter = objSht.FilterMode`
	`End Functiont.`

`Type`	Gibt den Arbeitsblatttyp zurück. Beispiel:
	`Sub ArbeitsblattTyp()`
	`  Select Case ThisWorkbook.Sheets("Tabelle1").Type`
	`  Case Excel.XlSheetType.xlWorksheet`
	`  MsgBox "Dies ist ein Arbeitsblatt."`
	`  Case Excel.XlSheetType.xlDialogSheet`
	`  MsgBox "Dies ist ein Dialogblatt."`
	`  Case Excel.XlSheetType.xlChart`
	`  MsgBox "Dies ist ein Diagrammblatt."`
	`  End Select`
	`End Sub`

1.4 Spezielle Eigenschaften des `Chart`-Objekts

Eigenschaft	Bedeutung
`ChartType`	Gibt den Diagrammtyp zurück oder legt ihn fest. `ActiveChart.ChartType = xlXYScatterLines`

`ChartTitle`	Gibt den Titel des angegebenen Diagramms zurück oder legt ihn fest.
	`With Charts("Chart1")`
	`  .HasTitle = True`
	`  .ChartTitle.Text = "Umsatz im ersten Quartal"`
	`End With`

`DataTable`	Gibt die Datentabelle eines Diagramms zurück. `ActiveChart.DataTable.Select`

1.5 Wichtige Methoden von `Worksheet`- und `Chart`-Objekts

Methode	Bedeutung
`Activate`	Das angegebene Arbeits- oder Diagrammblatt wird in den Vordergrund gebracht. ActiveWorkbook.Sheets(1).Activate
`Copy`	Das Blatt wird an eine andere Stelle der Arbeitsmappe kopiert. `ActiveSheet.Copy After:=Sheets(Sheets.Count)`

`Delete`	Löscht das angegebene Arbeits- oder Diagrammblatt.
	`Sub BlattLoeschen()`
	`  Application.DisplayAlerts = False`
	`  Sheets(4).Delete`
	`  Application.DisplayAlerts = True`
	`End Sub`

`Move`	Verschiebt das angegebene Arbeits-oder Diagrammblatt an eine andere Stelle in der Arbeitsmappe.
	`Sub BlattVerschieben()`
	`  Const conShtName As String = "VerschiebMich"`
	`  With ThisWorkbook`
	`  .Sheets(1).Name = conShtName`
	`  MsgBox "Pos. vorher: " & .Sheets(conShtName).Index`
	`  .Sheets(1).Move After:=.Worksheets(.Worksheets.Count)`
	`  MsgBox "Pos. nachher: " & Sheets(conShtName).Index`
	`  End With`
	`End Sub`

`Protect`	Schützt ein Arbeits-oder Diagrammblatt, damit es nicht geändert werden kann.
	`Sub BlattSchuetzen()`
	`  Const conPwd As String = "xyz_123"`
	`  ActiveSheet.Protect Password:=conPwd`
	`End Sub`

`SaveAs`	Speichert Änderungen am Diagramm oder Arbeitsblatt in einer anderen Datei. Siehe Prozedur `AktuellesBlattSpeicher`
`PrintOut`	Druckt das Arbeits- oder Diagrammblatt. `ActiveSheet.PrintOut`
`Unprotect`	Hebt den Blattschutz wieder auf (vgl. `Protect`) `ActiveSheet.Unprotect Password:=conPwd`

Die Anwendung der Methode `SaveAs` wird in der folgenden Prozedur mit dem Namen `AktuellesBlattSpeichernUnter` gezeigt. In der Prozedur wird die Excel-Version abgefragt, so dass sie für alle neueren Versionen ablauffähig ist.

```
Sub AktuellesBlattSpeichernUnter()
   ' In Anlehnung an Ron de Bruin, Use VBA SaveAs in Excel 2007-2010,
   ' Januar 2010, http://www.rondebruin.nl/saveas.htm
   ' Gültig für Excel 2000-2010
   Dim varFileName  As Variant   ' Dateityp
   Dim objWkbNew    As Workbook  ' Ziel-Arbeitsmappe
   Dim lngFormatVal As Long      ' Nummer des Dateityps
   ' Excel-Version prüfen
   If Val(Application.Version) < 9 Then Exit Sub
   If Val(Application.Version) < 12 Then
      ' Zeigt das integrierte Dialogfeld 'Speichern unter' an
      ' und gibt als Dateiformat Excel 2000-2003 vor.
      varFileName = Application.GetSaveAsFilename(InitialFileName:="", _
         FileFilter:="Excel Dateien (*.xls), *.xls", _
         Title:="Aktuelles Blatt in neuer Arbeitsmappe speichern.")
      If varFileName <> False Then
         ' Aktuelles Blatt in einer anderen Arbeitsmappe speichern
         ActiveSheet.Copy
         Set objWkbNew = ActiveWorkbook
         ' Das 2000-2003 Dateiformat xlWorkbookNormal zum Speichern angewandt
         objWkbNew.SaveAs varFileName, FileFormat:=-4143, CreateBackup:=False
         objWkbNew.Close False
         Set objWkbNew = Nothing
      End If
   Else
      ' Der Anwender hat die Wahl im alten 2000-2003 Dateiformat zu speichern
      ' oder in einem der neuen Excel-Dateiformate Standardwert für das
      ' Listenfeld 'Dateityp' ist Excel-Arbeitsmappe mit Makros (xlsm)
      varFileName = Application.GetSaveAsFilename(InitialFileName:="", _
            FileFilter:= _
            " Excel-Arbeitsmappe ohne Makros (*.xlsx), *.xlsx," & _
            " Excel-Arbeitsmappe mit Makros (*.xlsm), *.xlsm," & _
            " Excel 2000-2003 Format (*.xls), *.xls," & _
            " Excel-Binärarbeitsmappe (*.xlsb), *.xlsb", _
         FilterIndex:=2, Title:="Aktuelles Blatt in neuer Arbeitsmappe speichern.")
      ' Das richtige Dateiformat bestimmen für das Listenfeld 'Dateityp"
      If varFileName <> False Then
         Select Case LCase(Right(varFileName, Len(varFileName) - _
                        InStrRev(varFileName, ".", , 1)))
            Case "xls": lngFormatVal = 56
            Case "xlsx": lngFormatVal = 51
            Case "xlsm": lngFormatVal = 52
            Case "xlsb": lngFormatVal = 50
            Case Else: lngFormatVal = 0
         End Select
         ' Die Datei kann nun mit dem xlFileFormat-Parameter gespeichert werden,
         ' der mit dem Dateizusatz übereinstimmt.
         If lngFormatVal = 0 Then
            MsgBox "Fehler: unbekannter Dateizusatz"
         Else
            ' Kopiert das aktuelle Blatt in die neue Arbeitsmappe
            ActiveSheet.Copy
            Set objWkbNew = ActiveWorkbook
            ' Neue Arbeitsmappe mit dem gewählten Dateiformat speichern
            objWkbNew.SaveAs varFileName, _
               FileFormat:=lngFormatVal, CreateBackup:=False
            objWkbNew.Close False
            Set objWkbNew = Nothing
         End If
        End If
   End If
End Sub
```

Workshop 7: VBA-Programmierung mit MS Excel

1.6 Wichtige Eigenschaften und Methoden der `Sheets`-Auflistung

Eigenschaft/ Methode	Bedeutung
Count	Es wird die Anzahl der Blätter in der Auflistung angezeigt. `MsgBox ActiveWorkbook.Sheets.Count`

Add	Erzeugt ein neues Blatt vom Typ `xlWorksheet` oder vom Type `xlChart`.
	`Sub BlattHinzufuegen()`
	`  Sheets.Add after:=Sheets("Tabelle1"), Type:=xlWorksheet`
	`End Sub`

Select	Mit dieser Methode wird das angegebene das Objekt markiert. `ActiveWorkbook.Sheets("Tabelle2").Select`
Item	Diese Eigenschaft gibt ein einzelnes Blatt der Auflistung zurück. `Sheets.Item(3).Select`

1.7 Tabellenblätter auswählen

Um ein Tabellenblatt auszuwählen wird die Methode `Select` oder `Activate` angewandt. Allerdings bietet `Select` die Möglichkeit, mehrere Tabellenblätter auf einmal auszuwählen und diese dann gemeinsam als Gruppe zu bearbeiten:

```
Worksheets(Array("Tabelle1", "Tabelle3")).Select
```

Soll innerhalb dieser Gruppe das erste Tabellenblatt aktiviert werden, wird wie folgt codiert:

```
Worksheets(1).Activate
```

1.8 Neues Tabellenblatt hinzufügen

Mit der `Add`-Methode wird ein eines Blatt hinzugefügt. Steht in einer Programmzeile `Worksheets.Add`, wird, vor dem aktuellen Arbeitsblatt ein Tabellenblatt eingefügt. Über mehrere Parameter kann bewirkt werden, wie viele und wo die neuen Tabellenblätter hinzugefügt werden. Wird die allgemeinere Form `Sheets.Add` verwandt, kann über den Parameter `Type` festgelegt werden, ob ein Tabellen- oder Diagrammblatt in die Arbeitsmappe aufgenommen wird.

```
Sub NeuesBlattHinzufuegen()
   ' Tabellenverzeichnis hinzufügen
   Dim objSht As Worksheet ' Objektvariable
   Dim intRow As Integer   ' Zeilenzähler
   With ActiveWorkbook
      .Sheets(1).Select
      .Sheets.Add Type:= xlWorksheet
      .Sheets(1).Name = "Tabellenverzeichnis"
      intRow = 1
      For Each objSht In .Worksheets
         If objSht.Index > 1 Then
            Range("A" & intRow - 1).Value = objSht.Name
         End If
         intRow = intRow + 1
      Next objSht
   End With
End Sub
```

Mit der Anweisung

```
Worksheets.Add After:=Worksheets(Worksheets.Count)
```

wird ein neues Tabellenblatt hinter dem letzten hinzugefügt, mit

```
Worksheets.Add Before:=Worksheets(1)
```

hingegen vor dem ersten Tabellenblatt. Der Parameter `Count` wird verwandt, wenn mehr als ein neues Arbeitsblatt eingefügt werden soll. Soll einer Diagrammblatt hinzugefügt werden, schreibt man

```
Sheets.Add Before:=Worksheets(1), Type:=xlChart
```

oder einfach

```
Charts.Add Before:=Worksheets(1)
```

1.9 Tabellenblätter kopieren / verschieben

Tabellenblätter lassen sich kopieren und verschieben. Die allgemeine Syntax dafür ist fast gleich:

Ausdruck`.Copy(Before, After)`
Audruck`.Move(Before, After)`

Audruck ist dabei eine `Worksheets`- oder `Sheets`-Auflistung. Ebenso lassen sich `Chart`-Objekte kopieren und verschieben.

2.10 Tabellenblätter benennen

Beim Einfügen von Tabellenblättern werden sie von Excel selbstständig mit dem Namen `TabelleX` (wobei `X = 1, 2, 3, ...`) versehen. Diese Namen können jederzeit durch sinnvollere ersetzt werden.

Wird ein Tabellenblatt kopiert, so wird es unter dem Namen des ursprünglichen eingefügt und erhält einen zusätzlichen Zähler in Klammern. Mit

```
Worksheets(1).Copy After:=Worksheets(3)
```

wird das erste Tabellenblatt kopiert und hinter dem letzten eingefügt (siehe Abb. 2)

Abb. 2 Reihenfolge der Tabellenblätter nach dem Kopieren von Tabelle1

Die Umbenennung des kopierten Tabellenblatts erfolgt mit einer der folgenden Befehle:

```
ActiveSheet.Name = "Termine"
Worksheets("Tabelle1 (2)").Name = "Termine"
```

Um ein Tabellenblatt nach dem aktuellen Tagesdatum zu benennen, wird unter Verwendung der Funktion `Format` geschrieben:

```
ActiveSheet.Name = Format(Now, "dd. mmmm yyyy")
```

2.11 Tabellenblätter löschen

Ein Tabellenblatt wird mit der Methode `Delete` gelöscht:

```
Worksheets(1).Delete
Worksheets("Termine").Delete
```

Um die beim Löschen systemseitig ausgelöste Sicherheitsabfrage zu deaktivieren, kann mit `Application.DisplayAlerts = False` die Warnung unterdrückt werden.

2.12 Tabellenblätter aus- und einblenden

Mit der Eigenschaft `Visible` können Tabellenblätter ein- bzw. ausgeblendet werden. Das ist oft hilfreich, wenn auf dem unsichtbaren Blatt sogen. Hilfsrechnungen vorgenommen werden oder sonstige Daten gespeichert sind, die der Anwender nicht sehen soll. *Achtung:* Nicht alle Tabellenblätter können ausgeblendet werden. Eines muss mindestens sichtbar bleiben. Mit der Programmzeile

```
Worksheets(2).Visible = xlSheetHidden ' bzw. False
```

wird das zweite Tabellenblatt ausgeblendet. Mit der Programmzeile

```
Worksheets(2).Visible = xlSheetVisible ' bzw. True
```

wird es wieder angezeigt. Um zu verhindern, dass ein Anwender ein unsichtbares Tabellenblatt über die normale Oberfläche wieder sichtbar macht, kann die Konstante `xlVeryHidden` benutzt werden:

```
Worksheets(2).Visible = xlVeryHidden
```

In diesem Fall kann die ausgeblendete Tabelle nur mit Hilfe einer Prozedur wieder verfügbar gemacht werden.

2.13 Kopf- und Fußzeilen anlegen

Standardmäßig werden in Excel keine Kopf- und Fußzeilen ausgedruckt. Mit dem Objekt `PageSetup` kann man sich darum aber selbst kümmern.

```
Sub Kopf_Fuss_ZeileAnlegen()
   Worksheets("Tabelle1").Activate
   With ActiveSheet.PageSetup
      .LeftFooter = "Mappe: " & ActiveWorkbook.FullName
      .CenterFooter = "Uhrzeit: " & Time
      .RightFooter = "User: " & Application.UserName
     .LeftHeader = "&""Arial,Bold""&12" & "Zelle: " & ActiveSheet.Range("A1").Value
      .CenterHeader = "&""Arial,Bold""&12" & "Blatt: " & ActiveSheet.Name
      .RightHeader = "&""Arial,Bold""&12" & "Datum: " & Format(Date, "dd.mm.yyyy")
   End With
   ActiveWindow.SelectedSheets.PrintPreview
End Sub
```

Die in Abb. 3 gezeigte Kopf- und Fußzeile wurde mit der obigen Standardprozedur `Kopf_Fuss_ZeileAnlegen` erzeugt:

Zelle: A1	**Blatt: Tabelle1**	**Datum: 20.03.2011**
Mappe: Mappe1	Uhrzeit: 17:26:02	User: Volker

Abb. 3: Automatisch erzeugte Kopf- und Fußzeile für Tabelle1

Die in Abb. 4 gezeigte Fußzeile wurde mit der Standardprozedur `SetupPgFooter` erstellt.

Stand: 20.03.2011	Mappe: Mappe1	Seite: 1 / 1

Abb. 4: Automatisch erzeugte Fußzeile für alle Tabellenblätter der aktuellen Arbeitsmappe

```
Private Sub SetupPgFooter()
   Dim objSht As Worksheet
   For Each objSht In ActiveWorkbook.Worksheets
      If objSht.Range("A1") >= 1 Then
         With objSht.PageSetup
            .LeftFooter = "Stand: " + "&D"
            .CenterFooter = "Mappe: " & ActiveSheet.Parent.FullName
            .RightFooter = "&""Arial""&10Seite: &P / " + "&N"
         End With
      Else
         objSht.PageSetup.RightFooter = ""
      End If
   Next
End Sub
```

Mit neueren Excel-Versionen ab 2002 können Grafiken standardmäßig in Kopf- und Fußzeilen aufgenommen werden. Beispiel:

```
ActiveSheet.PageSetup.LeftHeaderPicture = "C:\Bilder\Logo.gif"
```

2.14 Tabellenblätter ausdrucken

Gedruckt werden können entweder ein oder mehrere Tabellenblätter, eine ganze Arbeitsmappe, ein Druckbereich oder eine Markierung:

Aufgabe	Programmzeile
Einzelne Tabelle drucken	`Sheets("Tabelle1").PrintOut`
Tabelle mit Kopien drucken	`Sheets("Tabelle1").PrintOut Copies:=3`
Ganze Arbeitsmappe drucken	`ActiveWorkbook.PrintOut`
Markierten Bereich drucken	`Selection.PrintOut Copies:=1, Collate:=True`

Wenn nicht direkt gedruckt werden soll, kann die Prozedur `DruckenDialogAufrufen` eingesetzt werden, die den eingebauten Drucken-Dialog aufruft:

```
Sub DruckenDialogAufrufen()
   Application.Dialogs(xlDialogPrint).Show
End Sub
```


Workshop 7: VBA-Programmierung mit MS Excel

2.15 Tabellenblatt als E-Mail versenden

Um das aktive Tabellenblatt als E-Mail-Anhang zu versenden, kann die Prozedur `TabellenblattAnhaengen` eingesetzt werden.

```
Sub TabellenblattAnhaengen()
   ' Tabellenblatt als E-Mail versenden
   Const conFile As String = "Anhang.xlsx"
   Dim strFullPath As String
   ' Vollständiger Pfad zum Anhang
   strFullPath = Application.DefaultFilePath & "\" & conFile
   If Val(Application.Version) < 12 Then
      strFullPath = Left(strFullPath, Len(strFullPath) - 1)
      Exit Sub
   End If
   Dim varEingabe As Variant
   varEingabe = InputBox(Prompt:="Empfänger der E-Mail", _
                         Title:="Tabellenblatt versenden", _
                         Default:="volker@dr-thormaehlen.de")
   If varEingabe <> vbNullString Then
      If Dir(strFullPath) <> vbNullString Then
         Kill strFullPath
      End If
      ' Aktuelles Tabellenblatt kopieren
      ActiveSheet.Copy
      ' Aktuelles Tabellenblatt speichern
      ActiveWorkbook.SaveAs strFullPath
      ' Empfängeradresse an den eingebauten Dialog übergeben
      Application.Dialogs(xlDialogSendMail).Show Arg1:=varEingabe
      ' Arbeitsmappe schließen
      Workbooks(conFile).Close
   End If
End Sub
```

Diese Prozedur erstellt folgende Nachricht in der E-Mail-Anwendung *OutLook*:

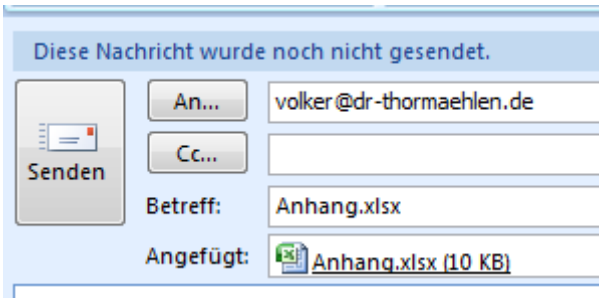

Abb. 3: Tabellenblatt als E-Mail-Anhang versenden

Workshop 7: VBA-Programmierung mit MS Excel

2 Ereignisse für das Tabellenblatt

2.1 Allgemeine Vorgehensweise

1. Die Tastenkombination Alt + F11 drücken, um in die Entwicklungsumgebung zu gelangen.
2. Im Projekt-Explorer einen Doppelklick auf eine Tabelle ausführen, der ein Ereignis zugeordnet werden soll.
3. Den Eintrag WORKSHEET im linken Kombinationsfeld des Code-Fensters auswählen.
4. Im rechten Kombinationsfeld das gewünschte Ereignis auswählen.

Für folgende 4 Ereignisse werden weiter unten zugehörige Ereignisprozeduren demonstriert:

Ereignis `(Worksheet_`	**Beschreibung (**Tritt ein, …
`BeforeDoubleClick`	Wird nach (!) einem Doppelklick auf eine Zelle aufgerufen.
`BeforeRightClick`	Wird nach (!) einem Klick mit der rechten Maustaste auf gerufen.
`Change`	wenn sich der Inhalt einer Zelle ändert.
`SelectionChange`	wenn sich die Markierung auf einem Tabellenblatt ändert.

2.2 Spaltenbreite automatisch anpassen

Mit dem Ereignis `Worksheet_Change` kann dafür gesorgt werden, dass direkt nach der Eingabe die Spaltenbreite automatisch angepasst wird. Die folgende Ereignisprozedur enthält den Code für dieses Ereignis:

```
Private Sub Worksheet_Change(ByVal Target As Range)
   Cells.EntireColumn.AutoFit
   Cells.EntireRow.AutoFit
End Sub
```

2.3 Tabellenname aus Zelle entnehmen

Im folgenden Codebeispiel wird die Zelle A1 zur Benennung des entsprechenden Tabellenblatts herangezogen. Jede Änderung des Inhalts von Zelle A1 bewirkt eine Änderung der Benennung des Tabellenblatts.

```
Private Sub Worksheet_Change(ByVal Target As Range)
   If Target.Address = "$A$1" And IsDate(Target.Value) Then
      ActiveSheet.Name = Format([A1].Value, "mmm yy")
   End If
End Sub
```

2.4 Aktive Zelle hervorheben

Die folgende Ereignisprozedur vergrößert die Schriftart und die Einfärbung der aktiven Zelle. Beim Verlassen der Zelle werden die ursprüngliche Einstellungen wieder hergestellt. Die Prozedur beruht auf dem Ereignis `Worksheet-SelectionChange`, das eintritt, wenn sich die Markierung auf dem aktiven Tabellenblatt ändert.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   With Cells
      .Interior.ColorIndex = xlColorIndexNone
      With .Font
         .FontStyle = "Regular"
         .Size = 10
      End With
   End With
   With Target
      .Interior.ColorIndex = 6
      With .Font
         .FontStyle = "Bold"
         .Size = 14
      End With
   End With
End Sub
```

2.5 Kontextmenü deaktivieren

Falls verhindert werden soll, dass das Zellenkontextmenü für einen bestimmten verbotenen Bereich (z. B. [A1:C5]) heruntergeklappt, wenn die rechte Maustaste gedrückt wird, dann hilft die folgende Ereignisprozedur:

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
   Dim rngTabu As Range
   Set rngTabu = ActiveSheet.Range("A1:C5")
   If Intersect(Target, rngTabu) Is Nothing Then
      Exit Sub
   Else
      Cancel = True
   End If
End Sub
```

2.6 Werterhöhung bei Doppelklick

Bei jedem Doppelklick auf die Zelle B1 soll deren Wert um 1 erhöht werden. Mit der folgenden Ereignisprozedur lässt sich das erreichen.

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
   If Target.Address = "$B$1" Then
      Target.Value = Target.Value + 1
      Cancel = True
  End If
End Sub
```

