Keine Angst vor Makros!

Word bringt eine eigene Entwicklungsumgebung für die hauseigene Programmiersprache *Visual Basic for Applications*, kurz VBA[1] mit. Damit lassen sich immer wieder gleiche Funktionsabläufe als so genannte »Makros« erstellen und bei Bedarf per Tastenkombination oder Schaltfläche abrufen.

Im Web gibt es auch zahlreiche Fertigmakros für bestimmte Aufgaben, die Sie herunterladen und in Ihre Dateien installieren können.

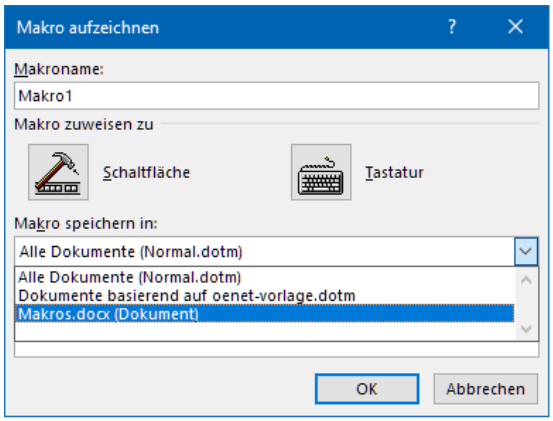
Makros aufzeichnen

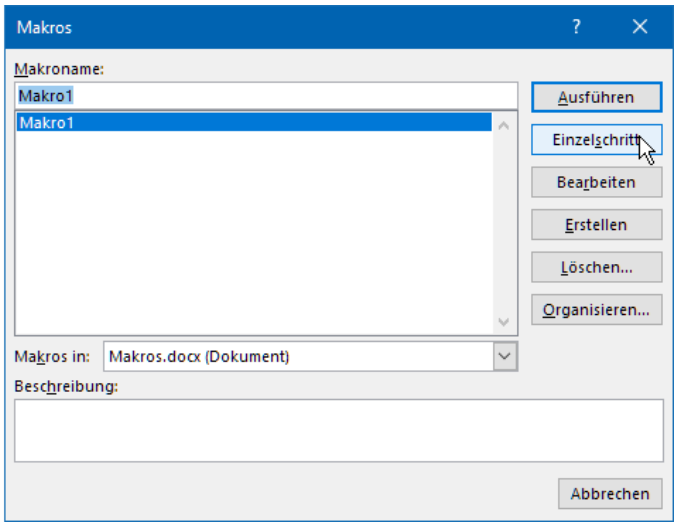
Mit geringem Aufwand lassen sich Vorgänge als Makro aufzeichnen und beliebig oft wiederholen.

Zum Aufzeichnen dient der Makrorekorder in der Registerkarte ENTWICKLERTOOLS | MAKRO AUFZEICHNEN. Den schnellen Zugriff erlaubt die Schaltfläche in der Statusleiste.

Zunächst müssen Sie bestimmen, ob das aufzuzeichnende Makro einer Schaltfläche oder einer Tastenkombination oder überhaupt nicht zugeordnet werden soll. Das Zuordnen können Sie jederzeit später noch nachholen.

Außerdem wird abgefragt, wo das Makro gespeichert werden soll. Beim Speichern in der Normal.dotm steht es Ihnen für alle Arbeiten mit Word zur Verfügung.





Ist der Makrorekorder mit Klick auf OK gestartet, nimmt der Mauszeiger die Form an und bis auf wenige Ausnahmen werden alle Aktionen aufgezeichnet.

---

[1] VBA ist keine einheitliche Programmiersprache für alle Office-Programme, sondern jedes Office-Programm verwendet einen eigenen »Dialekt«. Es ist darum nicht möglich, für ein Office-Programm entwickelte Makros in anderen Programmen zu verwenden.

Tipp

Mausaktivitäten sind bei der Makroaufzeichnung problematisch, weil häufig keine definierten Mausstandorte aufgezeichnet werden können oder weil sie gelegentlich ignoriert werden. Verwenden Sie beim Aufzeichnen die Tastatur F10 und ☰.

Nachdem Sie alle Schritte vollzogen haben, beenden Sie die Aufzeichnung mit ENTWICKLERTOOLS | AUFZEICHNUNG BEENDEN oder Klick auf die Schaltfläche ■ in der Statusleiste.

Alt+F8 öffnet die Liste der vorhandenen Makros. Rechts finden Sie Schaltflächen zum Starten oder Schritt-für-Schritt-Ausführen des links markierten Makros.

Mit BEARBEITEN können Sie sich anschauen, welcher Code bei der Aufzeichnung erzeugt wurde. Mit VBA-Kenntnissen lassen sich aufgezeichnete Makros nacharbeiten und erweitern.

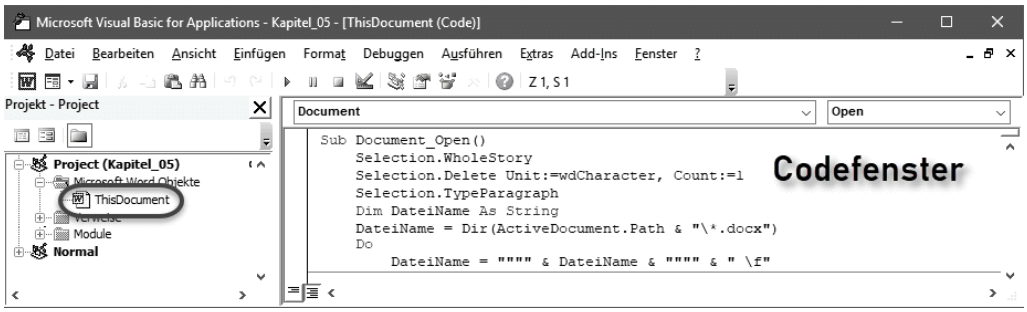
Im Bearbeitungsmodus lassen sich Makros mit F5 starten und mit F8 per Einzelschritt abarbeiten und prüfen.

Makros zum Download

Manche Leute stellen fertige Makros ins Internet, damit andere Leute die benutzen können. Eigentlich ein netter Zug, aber: Makros können auch Schaden anrichten. Microsoft hat es zu gut gemeint, die Programmiersprache »Visual Basic for Applications«, kurz VBA, kann auch auf das System zugreifen und dort ziemlich miese Dinge anstellen. Wenn Sie also Makros von irgendwoher beziehen, prüfen Sie, ob Sie dieser Quelle vertrauen können, bevor Sie deren Makros installieren und ausführen.

Makros installieren

Um ein Makro ins Dokument zu bringen, öffnen Sie mit ENTWICKLERTOOLS | VISUAL BASIC oder Alt+F11 den VBA-Editor; das ist die mitgelieferte Entwicklungsumgebung für diese Programmiersprache.



Die VBA-Entwicklungsumgebung

Kopieren Sie das Listing in das Codefenster des Editors. Achten Sie darauf, dass die Titelzeile *Sub Makroname()* und die Endzeile *End Sub* mitkopiert werden.

Wichtig!

Mit Makros ausgestattete Dateien müssen als Dateityp *Word Dokument mit Makros (*.docm)* gespeichert werden. Handelt es sich um Vorlagen, ist der Dateityp *Word Vorlage mit Makros (*.dotm)* die richtige Wahl.

Im Dialog MENÜBAND ANPASSEN haben Sie die Möglichkeit, den Makros in Ihrem Dokument eine Schaltfläche in einer Registerkarte oder eine Tastenkombination zuzuweisen.

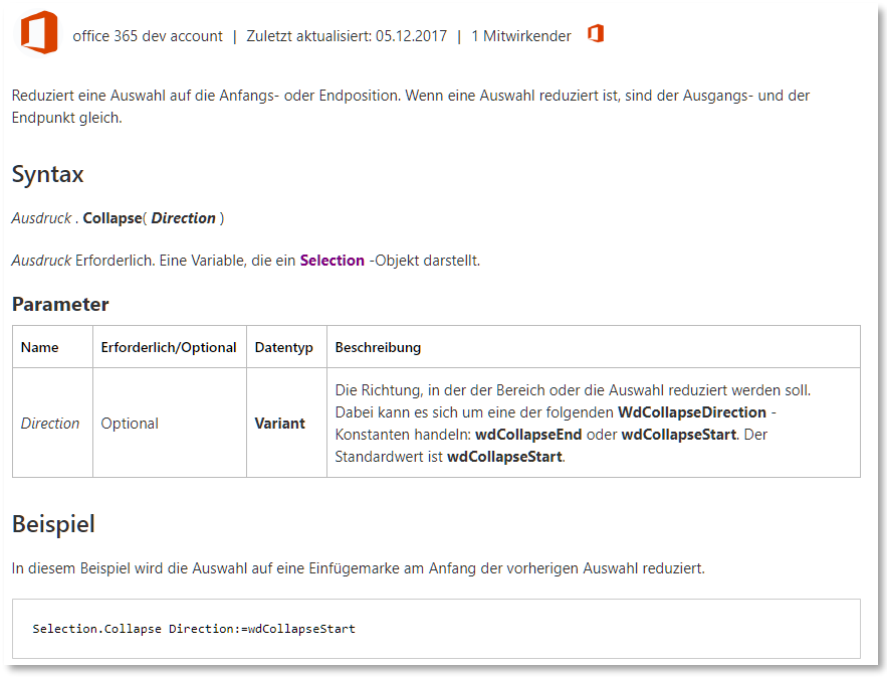
Makros modifizieren

Haben Sie ein Makro aufgezeichnet oder aus einer verlässlichen Quelle installiert, können Sie den VBA-Code natürlich auch modifizieren. Schlimmstenfalls funktioniert das Makro dann nicht mehr, was mit einer Fehlermeldung und Umschalten in den Debug-Modus quittiert wird. Der Debug-Modus ist nichts anderes als das oben schon erwähnte Editorfenster, und wenn Sie Glück haben, ist die Zeile, in der der Fehler aufgetreten ist, auch schon markiert. Das ist häufig nicht der Fall, und Sie können sich mit F8 durch den Code hangeln und zumindest die Zeile ermitteln, in der der Fehler auftritt.

Es gibt unendlich viele Fehlersituationen, und zum Programmieren gehört ein wenig Geschick und Grundkenntnisse der Computerlogik. Allerdings macht es Microsoft den Versuchswilligen leicht:

Zeichnen Sie ein Makro auf, wie oben beschrieben und öffnen Sie es mit Alt+F8 | BEARBEITEN.

Versuchen Sie zu verstehen, was da abläuft. Da Sie ja wissen, was Sie getan haben, ist die Zuordnung schon ein wenig erleichtert. Wenn Sie Begriffe nicht verstehen, geben Sie diese in eine Suchmaschine mit den Zusatz VBA. Microsoft bietet zu allen VBA-Befehlen Hilfeseiten an, auf denen die Funktionen, erforderliche Parameter und Syntax beschrieben, die meisten davon auch in deutscher Übersetzung.



office 365 dev account | Zuletzt aktualisiert: 05.12.2017 | 1 Mitwirkender

Reduziert eine Auswahl auf die Anfangs- oder Endposition. Wenn eine Auswahl reduziert ist, sind der Ausgangs- und der Endpunkt gleich.

Syntax

Ausdruck . **Collapse**(***Direction***)

Ausdruck Erforderlich. Eine Variable, die ein **Selection** -Objekt darstellt.

Parameter

Name	Erforderlich/Optional	Datentyp	Beschreibung
Direction	Optional	**Variant**	Die Richtung, in der der Bereich oder die Auswahl reduziert werden soll. Dabei kann es sich um eine der folgenden **WdCollapseDirection** -Konstanten handeln: **wdCollapseEnd** oder **wdCollapseStart**. Der Standardwert ist **wdCollapseStart**.

Beispiel

In diesem Beispiel wird die Auswahl auf eine Einfügemarke am Anfang der vorherigen Auswahl reduziert.

```
Selection.Collapse Direction:=wdCollapseStart
```

Seite mit Erläuterungen zu VBA-Befehlen

Beispiel

Angenommen, Sie benötigten immer wieder einen Ablauf mit folgenden Schritten:
Ein markiertes Wort soll fett gedruckt und in eckige Klammern gesetzt werden.
Im Folgenden erfahren Sie, wie Sie das Makro aufzeichnen und anschließend den Code verbessern.

1. Starten Sie die Aufzeichnung mit ⏺ und wählen Sie das aktuelle Dokument als Speicherort für das Makro.
2. Markieren Sie das zu bearbeitende Wort. (Wichtig: Für die Aufzeichnung macht es einen wesentlichen Unterschied, ob das Wort mit der Maus oder Tastatur markiert wird! Mehr dazu unten.)
3. Weisen Sie das Attribut »fett« mit Strg+⇧+F zu.
4. Bewegen Sie die Schreibmarke mit ← an den Anfang der Markierung.
5. Tippen Sie AltGr+7 für [.
6. Bewegen Sie die Schreibmarke mit Strg+→ ans Ende des Wortes.
7. Wenn das letzte markierte Zeichen der Leerschritt nach dem Wort ist, bewegen Sie die Schreibmarke mit ← einen Schritt zurück.
8. Heben Sie das Fett-Attribut wieder auf, indem Sie die Standardformatierung mit Strg+Leer wiederherstellen.
9. Tippen Sie AltGr+8 für].
10. Beenden Sie die Aufzeichnung mit Klick auf ■.
11. Öffnen Sie den Editor mit Alt+F8 | BEARBEITEN.

Dort sollte dieser [**Code**] zu sehen sein:

```
001  Sub Makro1()
002  '
003      Selection.Font.Bold = wdToggle
004      Selection.MoveLeft Unit:=wdCharacter, Count:=1
005      Selection.TypeText Text:="["
006      Selection.MoveRight Unit:=wdWord, Count:=1
007      Selection.MoveLeft Unit:=wdCharacter, Count:=1
008      Selection.Font.Reset
009      Selection.TypeText Text:="]"
010  End Sub
```

Code-Analyse:

Jeder Befehl in diesem Makro beginnt mit »`Selection.`«. `Selection` ist die Definition des Bereichs, auf den sich die nach dem Punkt folgende Aktion beziehen soll.

Mit Schritt 2 haben Sie das Wort markiert, aber diese Aktion wurde nur aufgezeichnet, wenn Sie das mit Strg, ⇧ und ← / → erledigt haben. Einen Doppelklick der Maus ignoriert der Makrorecorder. Das Makro arbeitet also nur dann völlig sicher, wenn das Wort vorab markiert wurde. Markierungen kann man aber auch durch das Programm vornehmen lassen, mehr dazu später.

Mehrfach tauchen Begriffe auf, die mit »`wd`« beginnen. Dieses Präfix kennzeichnet Word-Konstanten, mit denen die Wirkungsweise von bestimmten Befehlen beeinflusst werden kann, z. B.

`wdCharacter` zur Bestimmung der Textmenge, auf die sich ein Befehl bezieht. Alle Word-Konstanten sind auf dieser MSDN-Seite beschrieben.

```
001  Sub Makro1()
```

Der Makrotitel ist frei wählbar. Das Klammerpaar am Schluss dient zur Aufnahme von zu übergebenden Argumenten in »richtigen« VBA-Routinen.

```
002  '
```

Mit einem Apostroph beginnende Leerzeilen sind als Kommentarzeilen vorgesehen. Sie gehören nicht zum Code, sondern sind reine Erläuterungen zum Verstehen des Makros.
Mit einem Apostroph können Sie auch einer Codezeile einen Kommentar anhängen.
Ein Apostroph vor einer Codezeile sorgt dafür, dass die Zeile beim Ausführen des Makros übersprungen wird.

```
003      Selection.Font.Bold = wdToggle
```

Hier wird dem markierten Text das Fett-Attribut zugewiesen. Das `wdToggle` am Ende ist allerdings etwas Besonderes: Sollte die Markierung schon fett formatiert sein, nimmt dieser Befehl die Formatierung zurück, genauso wie die Schaltfläche F oder Strg+⇧+F. Um sicherzustellen, dass beim Ausführen des Makros immer das Fett-Attribut zugewiesen wird, ändern wir `wdToggle` in `True`, also `Selection.Font.Bold = True`

```
004      Selection.MoveLeft Unit:=wdCharacter, Count:=1
```

Die Schreibmarke wird an den Anfang der Markierung bewegt; zugleich wird die Markierung aufgehoben. Als Einheit ist zwar `Unit:=wdCharacter` angegeben, aber in diesem Fall zählt die gesamte Markierung als ein Zeichen. Der Befehl könnte auch
Selection.MoveLeft Unit:=wdWord, Count:=1
lauten oder ganz elegant
`Selection.Collapse Direction:=wdCollapseStart`
(Das ist der auf dem Screenshot der VBA-Hilfeseite erläuterte Befehl.)
Die Markierung wird aufgehoben und die Schreibmarke mit der `Direction`-Angabe auf den bisherigen Startpunkt gesetzt. Wollte man am Endpunkt weitermachen, müsste dort `Direction:=wdCollapseEnd` stehen.

```
005      Selection.TypeText Text:="["
```

Das ist leicht zu durchschauen, was hier passiert: An der Schreibmarkenposition (also vor dem eben fett formatierten Wort) wird das Zeichen [eingefügt. Einzufügender Text muss immer in geraden Anführungszeichen stehen.

```
006      Selection.MoveRight Unit:=wdWord, Count:=1
```

Nun geht es zum Ende des Wortes, die gleichzeitig mit → gedrückte Strg-Taste wird bei der Aufzeichnung in die Unit wdWord umgesetzt.

```
007      Selection.MoveLeft Unit:=wdCharacter, Count:=1
```

Während der Aufzeichnung sprang die Schreibmarke gleich hinter das das Wort abschließende Leerzeichen, wie in Word üblich, weshalb die Schreibmarke von Hand um ein Zeichen zurückgesetzt wurde. Was aber, wenn auf das Wort ein Satzzeichen oder eine Zeilen-/Absatzendemarke folgt? Wir benötigen eine Bedingung, die anhand des Zeichens links neben der Schreibmarke entscheidet, ob sie zurückgesetzt wird oder nicht.

Zum Auswerten des Zeichens links von der Schreibmarke muss dieses markiert sein, also zur »`Selection`« werden. Dazu bedienen wir uns folgenden Codes:

```
Selection.MoveLeft Unit:=wdCharacter, Count:=1, Extend:=wdExtend
```

Nichts Neues bis auf das letzte Argument `Extend:=wdExtend`: Damit wird erreicht, dass die Schreibmarke durch `.MoveLeft` nicht nur bewegt, sondern das Zeichen auch markiert wird.
Nun lässt sich diese `Selection` in einer Bedingung abfragen:

Hinweis

Ein Leerschritt, gefolgt von einem Unterstrich am Zeilenende signalisiert dem Programm, dass die nächste Zeile auch noch zum Befehl gehört. Damit lassen sich lange Programmzeilen besser lesbar aufteilen.

Apropos:

Zeilenende im Programmcode ist immer ein Absatzumbruch ↵! Automatische Zeilenumbrüche und auch manuell erzwungene mit ⇧+↵ haben in Programmcodes nichts verloren.

Erläuterung der Programmzeile:

```
If Selection = " " Then Selection.Collapse _
Direction:=wdCollapseStart Else Selection.Collapse Direction:=wdCollapseEnd
```

Wenn das markierte Zeichen ein Leerzeichen " " ist, soll die Markierung aufgehoben und die Schreibmarke an deren Startpunkt gesetzt werden, also links vom Leerzeichen; anderenfalls soll die Markierung ebenfalls aufgehoben, aber die Schreibmarke ans Ende gesetzt werden. So wird sichergestellt, dass die Schreibmarke am Ende des Wortes steht, bevor der nächste Befehl ausgeführt wird.

```
008         Selection.Font.Reset
```

Die Zeile ist nötig, um die schließende Klammer nicht auch mit dem Fett-Attribut zu versehen. Die Formatierung wird auf die normale Formatierung des Textes zurückgesetzt.

```
009         Selection.TypeText Text:="]"
```

Hier passiert dasselbe wie in Zeile 005, nur diesmal mit der schließenden Klammer.

```
010     End Sub
```

Jedes Makro wird mit `End Sub` abgeschlossen.

Der überarbeitete Programmcode (1. Stufe):

Die vorab beschriebenen Änderungen wurden eingearbeitet.

```
001  Sub FettUndEckigeKlammern()
002      Selection.Font.Bold = True
003      Selection.Collapse Direction:=wdCollapseStart
004      Selection.TypeText Text:="["
005      Selection.MoveRight Unit:=wdWord, Count:=1
006      Selection.MoveLeft Unit:=wdCharacter, Count:=1, Extend:=wdExtend
007      If Selection = " " Then
008          Selection.Collapse Direction:=wdCollapseStart
009      Else
010          Selection.Collapse Direction:=wdCollapseEnd
011      End If
012      Selection.Font.Reset
013      Selection.TypeText Text:="]"
014  End Sub
```

Erläuterungen zu den Änderungen in den Zeilen 007 bis 011:

Wenn eine Bedingungsabfrage länger wird, empfiehlt es sich, sie der Übersicht halber auf mehrere Zeilen zu verteilen. »`If`« und »`Else`« stehen dann in separaten Zeilen. Die Bedingung wird mit der Zeile »`End If`« abgeschlossen.

Nächster Schritt: Das Wort automatisch markieren

Manko des aufgezeichneten Makros war ja, dass das Markieren per Mausklick nicht mit aufgezeichnet wurde. Um sicherzustellen, dass das Wort, in dem die Schreibmarke steht, wirklich markiert ist, übertragen wir die Aufgabe dem Programm.

Dazu bedarf es noch einmal der Betrachtung des Begriffs `Selection`. Auch wenn nichts markiert ist, existiert eine `Selection`, dann allerdings mit der Länge Null. Das lässt sich abfragen, denn Word erfasst den Beginn und das Ende der Markierung, darum gibt die Abfrage

`If Selection.Start = Selection.End`

Auskunft darüber, ob etwas markiert wurde oder nicht. Sind beide Positionen unterschiedlich, ist alles in Ordnung und nichts zu veranlassen. Bei Gleichheit allerdings ist die Markierung nachzuholen, deshalb

```
001  If Selection.Start = Selection.End Then
002     Selection.MoveLeft Unit:=wdWord, Count:=1
003     Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
004  End If
```

In Zeile 001 wird geprüft, ob nichts markiert ist, wenn ja, wird mit Zeile 002 die Schreibmarke an den Wortanfang gerückt und mit Zeile 003 das Wort bis zum Ende markiert.
Wenn im Nichtzutreffensfall nichts zu veranlassen ist, braucht es kein `Else`, es folgt sofort `End If`.

Dies gleich als Eingangsprüfung eingearbeitet in das überarbeitete Listing, ergibt dann diesen Code:

```
001  Sub FettUndEckigeKlammern()
002      If Selection.Start = Selection.End Then
003          Selection.MoveLeft Unit:=wdWord, Count:=1
004          Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
005      End If
006      Selection.Font.Bold = True
007      Selection.Collapse Direction:=wdCollapseStart
008      Selection.TypeText Text:="["
009      Selection.MoveRight Unit:=wdWord, Count:=1
010      Selection.MoveLeft Unit:=wdCharacter, Count:=1, Extend:=wdExtend
011      If Selection = " " Then
012          Selection.Collapse Direction:=wdCollapseStart
013      Else
014          Selection.Collapse Direction:=wdCollapseEnd
015      End If
016      Selection.Font.Reset
017      Selection.TypeText Text:="]"
018  End Sub
```

Nächster Schritt: Vereinfachung durch »With«

Programmierer sind schreibfaul. Um Schreibarbeit zu reduzieren, gibt es in VBA die Möglichkeit, Befehle, die sich immer wieder auf denselben *Ausdruck* beziehen, verkürzt darzustellen. Der sich hier ständig wiederholende Ausdruck ist Selection. Stellt man dem Code ein With Selection voran, beziehen sich alle nachfolgenden Befehle auf Selection, weshalb dieser Ausdruck solange nicht genannt werden muss, bis ein En With diese Definition aufhebt.

Also dann:

```
001  Sub FettUndEckigeKlammern()
002  With Selection
003      If .Start = .End Then
004          .MoveLeft Unit:=wdWord, Count:=1
005          .MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
006      End If
```

```
007         .Font.Bold = True
008         .Collapse Direction:=wdCollapseStart
009         .TypeText Text:="["
010         .MoveRight Unit:=wdWord, Count:=1
011         .MoveLeft Unit:=wdCharacter, Count:=1, Extend:=wdExtend
012         If Selection = " " Then
013             .Collapse Direction:=wdCollapseStart
014         Else
015             .Collapse Direction:=wdCollapseEnd
016         End If
017         .Font.Reset
018         .TypeText Text:="]"
019     End With
020     End Sub
```

Lediglich bei der Abfrage in Zeile 012 muss der Ausdruck noch einmal erwähnt werden.

Noch was: Wenn das Programm Text benutzt

In dem Beispielcode sind ja schon mehrfach Texte verarbeitet worden: die Klammern, die eingefügt wurden, der Leerschritt zum Prüfen. Wichtig ist immer das Einfassen des Textes in gerade Anführungszeichen. So lassen sich ganze Sätze ins Programm einfügen.

`Selection.TypeText Text:="Das soll im Text erscheinen."`
sorgt dafür, dass das Programm an der Schreibmarkenposition diesen Satz einfügt oder, sofern eine Markierung besteht, den markierten Text mit diesem Satz überschreibt.

Wird im Text ein Anführungszeichen benötigt, bedarf es eines doppelten Anführungszeichens im Textstring. `""""` steht also für ein einsames Anführungszeichen, `""` für einen leeren Text.

Zwischen die Anführungszeichen lassen sich alle per Tastatur oder mit Alt+ASCII-Code aufrufbaren Zeichen einfügen. Alternativ versteht VBA auch die direkte Angabe des ASCII-Codes, indem Sie diesen mit `Chr(#)` angeben, wobei # für den Code steht. Das ermöglicht auch das Eingeben von Steuerzeichen, z. B. `Chr(13)` für einen Absatzumbruch. Um einzelne Leerzeichen im Listing deutlicher zu erkennen, können Sie `" "` durch `Chr(32)` ersetzen.

Mehrere Textteile verbinden Sie dabei durch das &-Zeichen:
`"Dies ist eine Zeile," & Chr(13) & "auf die nach einem Umbruch diese folgt."`
`"&"` innerhalb eines Textstrings, also noch bevor ein zweites Anführungszeichen den String abschließt, wird als gewöhnliches Zeichen behandelt.

Fazit

- Mit dem Aufzeichnen von Makros eröffnet sich ein Weg, VBA praktisch zu erlernen.
- Nicht alles wird aufgezeichnet.
- Funktionen sind bei Microsoft MSDN dokumentiert.
- Überarbeitungen können aus einem Makro ein »intelligentes« Stück Programm machen.