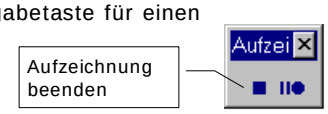
# Makroprogrammierung


Ein Makro ist in der einfachsten Form einfach eine aufgezeichnete Tasten- und Befehlsfolge, kann aber auch ein beliebig komplexes Visual Basic-Programm sein. Programmierung ist ein außerordentlich weites und komplexes Thema, und kann sicherlich nicht in ein paar Stunden erlernt werden. Wir werden uns an dieser Stelle nur ein paar einfache Grundlagen ansehen.

## Aufgabe 1: Makro aufzeichnen

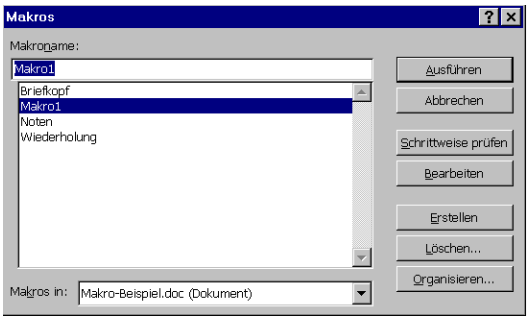
1.) Starten Sie Word
2.) Speichern Sie anschließend gleich die "leere" Datei z.B. unter dem Namen '**Makro-Beispiel**'.
3.) Wählen Sie Menüpunkt 'Extras', 'Makro', 'Aufzeichnen'
4.) Benennen Sie das Makro (oder lassen Sie den voreingestellten Namen 'Makro1').
5.) Wichtig: Ändern Sie im Feld 'Makro speichern:' von 'Alle Dateien' auf 'Makro-Beispiel' (damit das Makro *nur* in Ihrem eigenen, gerade geöffneten Dokument gespeichert wird!!).
6.) Klicken Sie auf 'OK'
    Ab nun werden alle Tastendrücke und alle Befehle aufgezeichnet! (das erkennen Sie auch am Mauszeiger)
7.) Schreiben Sie ein paar Wörter, dann Eingabetaste für einen neuen Absatz
8.) Beenden Sie die Aufzeichnung




## Aufgabe 2: Makro ausführen

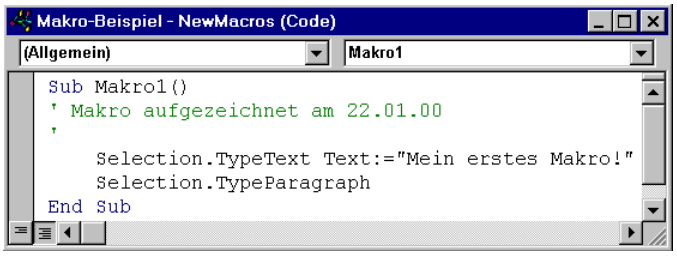
1.) Wählen Sie Menüpunkt 'Extras', 'Makro', 'Makros...', markieren Sie in der Liste das eben erstellte Makro, und klicken Sie auf 'Ausführen'.

Das Makro wiederholt jetzt alle aufgezeichneten Tastendrücke und Befehle, alle Vertipper und Korrekturen, und der Satz steht ein zweites mal da, genauso, als hätten Sie ihn nochmals eingetippt. Sie können das Makro nun beliebig oft aufrufen.

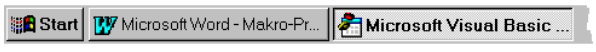



## Aufgabe 3: Makro bearbeiten

1.) Wählen Sie Menüpunkt 'Extras', 'Makro', 'Makros...', markieren Sie in der Liste das eben erstellte Makro, und klicken Sie auf 'Bearbeiten'.
2.) Im sogenannten Visual Basic-Editor können wir uns das "Innenleben" des Makro ansehen (Quellprogramm, source code).
    Jedes Makro ist eigentlich ein Unterprogramm (subroutine), darum beginnt es immer mir SUB und endet mit END SUB.




    Die Zeilen mit dem einfachen Anführungszeichen am Zeilenbeginn nennt man Kommentare. Kommentare dienen nur der Lesbarkeit durch den Menschen, sie werden bei der Makroausführung ignoriert.
    Ein spezieller Befehl ist `Selection.TypeText`. Wie wir vermuten können, ist es der Befehl, der Word anweist, den in `Text:=".."` angegebenen Text "einzutippen".
    Der Befehl `Selection.TypeParagraph` erzeugt offensichtlich einen Absatz (paragraph).
3.) Nachdem wir das Makro einigermaßen verstanden haben, getrauen wir uns vorsichtig an Änderungen:
    - ändern Sie den Text im Satz (z.B. 'erstes' auf 'zweites')
    - verdoppeln Sie den Befehl `Selection.TypeParagraph` (mittels Zwischenablage natürlich)
4.) Klicken Sie dann in der Taskleiste auf Word, und führen Sie das Makro erneut aus, und beobachten Sie, ob Sie das Makro tatsächlich erfolgreich geändert haben.



### Aufgabe 4: Benutzereingabe

1.) Zeichnen Sie ein neues Makro namens 'Benutzereingabe' auf, zeichnen Sie die Eingabe "Sg. Benutzer, Sie haben Folgendes eingegeben: " auf, beenden Sie die Aufzeichnung, und ergänzen Sie das Makro folgendermaßen:

```
Sub Benutzereingabe()
    was_der_Benutzer_eingegeben_hat = InputBox("Bitte geben Sie irgendwas ein")
    Selection.TypeParagraph
    Selection.TypeText Text:="Sg. Benutzer, Sie haben Folgendes eingegeben: "
    Selection.TypeText Text:=was_der_Benutzer_eingegeben_hat
    Selection.TypeParagraph
End Sub
```

2.) Wenn Sie das Makro ablaufen lassen, bewirkt der Befehl 'InputBox', dass ein kleines Fenster aufgeht, das dem Benutzer eine Eingabe ermöglicht. Damit der Benutzer auch weiß, was er eingeben soll, gibt man dem Befehl 'InputBox' einen Erklärungstext mit, der angezeigt wird.
Die Eingabe des Benutzers wird in obigem Beispiel in der Variablen 'was_der_Benutzer_eingegeben_hat' gespeichert (Wertzuweisung an die Variable links vom Gleichheitszeichen!). Sie als Programmierer/in können Variablen bekanntlich benennen wie Sie wollen. Sinnvoll sind sogenannte "sprechende" Namen, aus denen klar und deutlich hervorgeht, welche Inhalte diese Variable aufnimmt. Ein Variablenname 'was_der_Benutzer_eingegeben_hat' ist zwar sehr aussagekräftig, aber wohl auch ein bisschen zu lang und daher tippfehleranfällig, was aber wieder nicht so gut ist. Anbieten würden sich hier Namen wie 'benutzereingabe' oder 'eingabe' oder etwas in der Art.

### Aufgabe 5: Verzweigung

1.) Zeichnen Sie ein neues Makro namens 'Verzweigung' auf, und ergänzen Sie das Makro nach nebenstehendem Beispiel.
2.) Testen Sie das Makro, indem Sie der Variablen 'wochentag' verschiedene Werte zuweisen: In Abhängigkeit vom Inhalt der Variablen werden verschiedene Zweige im Programm durchlaufen!

```
Sub Verzweigung()

wochentag = "Mo"

If ( wochentag = "Sa" Or wochentag = "So" ) Then
      Selection.TypeText Text:="Wochenende, lange schlafen!"
   Else
      Selection.TypeText Text:="Aufstehen, Arbeiten gehen ..."
End If

Selection.TypeParagraph
End Sub
```

### Aufgabe 6: FOR-Schleife (Wiederholungen)

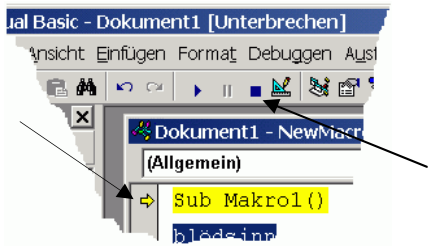
1.) Zeichnen Sie ein neues Makro namens 'Wiederholungen' auf, tippen Sie "Anfang", Absatz ↵, "Ich soll nicht schwätzen", Absatz ↵, "Ende". Vielleicht musste jemand von uns in fernen Volksschulzeiten diesen oder einen ähnlichen Satz 100 mal schreiben. Eintönige, gleichförmige Arbeiten sind aber das ideale Betätigungsfeld für Maschinen, daher werden diese Aufgabe dem Computer übertragen:

```
Sub Wiederholungen()

Selection.TypeText Text:="Anfang"
Selection.TypeParagraph

For i = 1 To 10
    Selection.TypeText Text:="Ich soll nicht schwätzen"
    Selection.TypeParagraph
    Next i

Selection.TypeText Text:="Ende"
End Sub
```

2.) Bearbeiten Sie das Makro folgendermaßen:
Der Befehl FOR *Laufvariable* = *Anfangswert* TO *Endwert* ... NEXT dient zur Formulierung von Programmschleifen (Wiederholungen).
Am Beginn der FOR-Schleife wird einer Variablen ein Anfangswert zugewiesen (eine Variable ist ein Speicherplatz, z.B. für eine einzelne Zahl). Im Beispiel hat die Laufvariable der FOR-Schleife den Namen 'i' und bekommt den Anfangswert 1 zugewiesen.
Dann wird der (eingerückt dargestellte) Block von Anweisungen, bis zur NEXT-Anweisung, ausgeführt. Anschließend wird der Wert der Laufvariablen i um 1 erhöht, und mit dem Endwert (10) verglichen. Wenn der Wert von i kleiner oder gleich 10 ist, wird die Schleife erneut ausgeführt.
Wenn der Wert von i größer 10 ist, wird die Schleife verlassen, und die Programmausführung wird nach NEXT fortgesetzt.
3.) Führen Sie das Makro aus.

Eine FOR-Schleife wird dann verwendet, wenn die Anzahl der Schleifendurchläufe von vornherein feststeht.
Es gibt aber noch andere Schleifentypen, auf die wir jetzt nicht eingehen.

Beim Programmieren können dem/der Programmierer/in logische Fehler und syntaktische Fehler unterlaufen. Syntax-Fehler sind ein Verstoß gegen "Rechtschreibung und Grammatik" der Programmiersprache. Logische Fehler sind z.B. Denkfehler in der Ablauflogik, oder wenn der/die Programmierer/in nicht alle Möglichkeiten bedenkt.
Wichtig: Keine Programmiersprache kann logische Fehler erkennen; wenn man Unsinn programmiert, macht das Programm genau den programmierten Unsinn.
Syntax-Fehler entstehen, wenn wir unzulässige Sprachkonstruktionen verwenden, oder uns einfach vertippt haben. In diesem Fall wird der Programmablauf unterbrochen (gelbes Pfeilchen). Nach der Korrektur des Fehlers müssen wir das unterbrochene Makro stoppen (blaues Viereck), um es dann erneut starten zu können.
Auf Englisch werden Programmfehler bugs (Ungeziefer) genannt, Fehlersuche nennt man daher debugging.



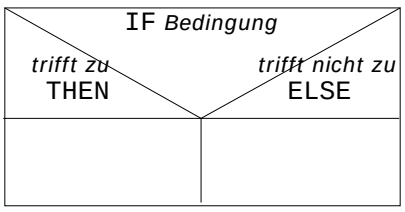

## Aufgabe 7: Erweiterung des Wiederholungen-Makros um eine Benutzereingabe

1.) Wir möchten nun das Programm flexibler machen. Es soll die Anzahl der Schleifendurchläufe (10) nicht fest im Programmcode stehen, wo nur ein Programmierkundiger etwas ändern kann, sondern ein ganz harmloser Word-Benutzer soll die Anzahl eingeben können:
Wenn die Funktion `InputBox("..")` ausgeführt wird, hält die Programmausführung an, und ein Fenster erscheint mit der Aufforderung, die Anzahl einzugeben. Natürlich muss die Benutzereingabe irgendwo gespeichert werden, und dafür führen wir einfach eine Variable namens 'antwort' ein.
Dann ändern wir die FOR-Schleife, sodass nicht wie bisher immer bis 10, sondern flexibel soweit gezählt wird, wie es dem Wert der Variablen 'antwort' entspricht: `For i = 1 To antwort`

```
Sub Wiederholungen()
'
Selection.TypeText Text:="Anfang"
Selection.TypeParagraph

antwort = InputBox("Schleifendurchlauf wie oft ?")

For i = 1 To antwort
    Selection.TypeText Text:=i
    Selection.TypeText Text:=".) Ich soll nicht schwätzen"
    Selection.TypeParagraph
    Next i

Selection.TypeText Text:="Ende"
End Sub
```

2.) Weiters möchten wir zusehen, wie sich der Wert der Variablen i im Programmablauf ändert, deshalb lassen wir den Inhalt der Variablen i in jeder Zeile ausgeben: `Selection.TypeText Text:=i` (natürlich darf diesmal `i` nicht in Hochkommas stehen (`"i"`), sonst würde ja nicht der Inhalt der Variablen `i`, sondern einfach der Buchstabe i ausgegeben).

## Aufgabe 8: Anwendungsbeispiel 'Pruefung'




Probieren Sie das Makro aus.
Beachten Sie wieder: `"Note"` sind die 4 Buchstaben N, o, t, e, während `note` den Inhalt der Variablen note meint.

```
Sub Pruefung()
'
antwort = InputBox("Ihre Note ?")
Selection.TypeParagraph
If note < 5 Then
        Selection.TypeText Text:="Note "
        Selection.TypeText Text:=note
        Selection.TypeText Text:=" ist positiv"
    Else
        Selection.TypeText Text:="Note "
        Selection.TypeText Text:=note
        Selection.TypeText Text:=" ist leider negativ"
    End If
End Sub
```

Leider ist dem Programmierer bei nebenstehendem Beispiel ein logischer Fehler unterlaufen. Finden Sie den Fehler, und korrigieren Sie ihn!! (Schwierig? Hinweise: Lesen Sie nochmals zum Thema 'Wertzuweisungen an Variable' nach. Was würden Sie erwarten, dass die Zeile `Selection.TypeText Text:=note` tun sollte? Warum tut sie das nicht, was Sie erwarten? Woher bekommt die Variable `note` ihren Inhalt her?)

Damit haben Sie alle wichtigen Elemente von Programmiersprachen kennengelernt. Natürlich ergeben diese Winzig-Beispiele für sich alleinstehend noch keinen Sinn. Richtige Programme bestehen aus hunderten, tausenden Befehlen, ineinander geschachtelten Schleifen und Verzweigungen. Das erfordert aber viel Übung, und geht über den Rahmen unserer Übungen hinaus.

Mit den Programmierkenntnissen von nur einer Stunde ist es schwierig, etwas Sinnvolles zu programmieren. Es dauert normalerweise Wochen und Monate, bis man genug Wissen und Übung hat, um größere, nützliche Programme erstellen zu können. Ein kleines Beispiel dafür, dass man auch mit relativ geringem Aufwand etwas Nützliches bauen kann, finden Sie in folgender Übung:

### Aufgabe 9: ein Anwendungsbeispiel: Schriftarten

Dieses Makro listet alle auf diesem PC installierten Schriftarten auf.
Zeichnen Sie ein neues Makro auf, drücken Sie die Tabulator-Taste, beenden Sie die Aufzeichnung, und bearbeiten und ergänzen Sie das Makro folgendermaßen:

Die Funktion `WordBasic.CountFonts()` liefert die Anzahl der installieren Schriften zurück.
In einer `for` -Schleife werden alle Schriftartnamen ausgelesen (`WordBasic.[Font$](i)` gibt den Namen der i-ten Schriftart zurück), der Schriftartname ausgegeben (`Selection.TypeText`), die Schriftart eingestellt (`WordBasic.Font`), und eine Beispieltext ausgegeben.

```
Sub Schriftarten()
'

WordBasic.FileNew

WordBasic.Font "Arial", 12
Selection.TypeText Text:="Alle " & WordBasic.CountFonts() & " auf diesem PC
installierte Schriftarten:"
Selection.TypeParagraph
Selection.TypeParagraph

For i = 1 To WordBasic.CountFonts()
    WordBasic.Font "Arial", 10
    Selection.TypeText Text:=i
    Selection.TypeText Text:=".) "
    Schriftartname = WordBasic.[Font$](i)
    Selection.TypeText Text:=Schriftartname + ":"
    Selection.TypeText Text:=vbTab
    WordBasic.Font WordBasic.[Font$](i), 10
    Selection.TypeText Text:="das ist ein Beispiel in der gewählten Schriftart"
    Selection.TypeParagraph
    Next i

WordBasic.Font "Times New Roman", 10

End Sub
```

Alle 97 auf diesem PC installierte Schriftarten:

| | |
|---|---|
| Arial Unicode MS: | das ist ein Beispiel in der gewählten Schriftart |
| Arial Black: | **das ist ein Beispiel in der gewählten Schriftart** |
| Arial Narrow: | das ist ein Beispiel in der gewählten Schriftart |
| Book Antiqua: | das ist ein Beispiel in der gewählten Schriftart |
| Century Gothic: | das ist ein Beispiel in der gewählten Schriftart |
| Comic Sans MS: | das ist ein Beispiel in der gewählten Schriftart |
| Courier New: | das ist ein Beispiel in der gewählten Schriftart |
| Haettenschweiler: | das ist ein Beispiel in der gewählten Schriftart |
| Impact: | das ist ein Beispiel in der gewählten Schriftart |
| Monotype Corsiva: | *das ist ein Beispiel in der gewählten Schriftart* |
| Symbol: | δασ ιστ εινΒειστιελ ιν δερ γεω™ηλτεν Σχηριφταρτ |
| Times New Roman: | das ist ein Beispiel in der gewählten Schriftart |
| Wingdings: | ♎♋♦ ♓♦◆ ♏♓■ ✍♏♓♦❑♓♏● ♓■ ♎♏❒ ♑♏⬥ |

Weiterführendes über Makro-Programmierung mit Excel:
"EXCEL schneller, rascher, sicherer mit Visual Basic für Anwendungen" von Josef Broukal
http://www.frank-moehle.de/computing/literatur/VBASeminar/JOANNEUM%20RESEARCH_VBA_Seminar.html