P r o g r a m m a b l a u f s t e u e r u n g

Jede Programmiersprache braucht Konstrukte zur Steuerung des Programmablaufs. Grundsätzlich unterscheiden wir ***Verzweigungen*** und ***Schleifen***. Schleifen dienen dazu, bestimmte Anweisungen wiederholt auszuführen, während Verzeigungen entweder den einen oder anderen Anweisungsblock auszuführen, je nachdem, ob eine Bedingung erfüllt ist oder nicht.

1. Verzweigungen

Bei Verzweigungen unterscheidet man im Allgemeinen zwischen einfachen Verzweigungen und Mehrfachverzweigungen oder Mehrfachauswahlen.

If-Anweisung

Die *If*-Anweisung kommt in VBA in vielen verschiedenen Varianten vor. Sie steuert den Programmablauf dadurch, dass bestimmte Anweisungen nur dann ausgeführt werden, wenn die formulierte Bedingung wahr ist. Die einfachste Form ist die einzeilige Variante.

BSP_IFTHEN

```
Private Sub BSP_IFTHEN_Click()
   Dim X As Integer
   X = InputBox("Bitte eine ganze Zahl eingeben!", "EINGABE", "0")
   If X > 0 Then MsgBox "Zahl > 0"
End Sub
```

Dieses Beispiel liest zunächst über den Eingabedialog *Inputbox* eine Zahl ein. Die *If*-Anweisung überprüft dann, ob der Wert von X größer als 0 ist. Wenn die Bedingung wahr ist, wird in einer Dialogbox mit *MsgBox* die Meldung *Zahl > 0* ausgegeben.

Diese Form der *If*-Anweisung hat den Nachteil, dass bei Zutreffen der Bedingung nur genau eine Anweisung ausgeführt werden kann. Für den Fall, dass Sie mehrere Anweisungen beim Zutreffen der Bedingung ausführen möchten, stellt VBA auch eine mehrzeilige Variante zur Verfügung.

BSP_IFTHEN2

```
Private Sub BSP_IFTHEN2_Click()
   Dim X As Integer
   X = InputBox("Bitte eine ganze Zahl eingeben!", "EINGABE", "0")
        If X > 0 Then
                MsgBox "Zahl > 0"
                'hier könnten weitere Anweisungen stehen
        End If
End Sub
```

Bei dieser Form muss die *If*-Verzweigung durch die Zeile *End If* abgeschlossen werden, damit VBA erkennt, welche Zeilen noch zum *Then*-Zweig gehören und welche nach Verlassen der *If*-Anweisung ausgeführt werden sollen.

BSP_IFTHEN3

```
Private Sub BSP_IFTHEN3_Click()
   Dim X As Integer
   X = InputBox("Bitte eine ganze Zahl eingeben!", "EINGABE", "0")
        If X > 0 Then
                MsgBox "Zahl > 0"
                If X > 10 Then
                        MsgBox "Zahl ist größer als 10"
                End If
        End If
End Sub
```

Meistens möchten Sie aber nicht nur bestimmen, was geschehen soll, wenn die Bedingung erfüllt ist, sondern auch, welche Anweisung ausgeführt werden soll, wenn die Bedingung **nicht** erfüllt ist. Für diese Fälle stellt VBA die *If-Then-Else*-Anweisung zur Verfügung.

BSP_IFTHENELSE

```
Private Sub BSP_IFTHENELSE_Click()
   Dim X As Integer
   X = InputBox("Bitte eine ganze Zahl eingeben!", "EINGABE", "0")
        If X > 0 Then
                MsgBox "Zahl > 0"
        Else
                If X = 0 Then
                        MsgBox "Zahl = 0"
                Else
                        MsgBox „Zahl < 0“
                End If
        End If
End Sub
```

Hier wird nun auch nach dem Einlesen des Wertes X zunächst überprüft, ob X > 0 ist. Falls dies der Fall ist, wird wie in den anderen Beispielen eine entsprechende Meldung ausgegeben. Im anderen Fall wird im *Else*-Zweig in einer weiteren *If-Then-Else*-Anweisung überprüft , ob die Zahl gleich 0 ist oder kleiner 0 ist und die entsprechende Meldung ausgegeben. Sie sehen hier, dass auch der *Else*-Zweig weitere *If*-Anweisungen oder Schleifen oder sonstige Anweisungen enthalten kann.

ACHTUNG:

Zur besseren optischen Erfassung der Prozedurinhalte sollten Sie den Inhalt von *If*-Anweisungen, aber auch von Schleifen sinnvoll einrücken. Wenn Sie dann mehrere *If*-Anweisungen ineinander verschachteln, behalten Sie den Überblick, welche *End If* - Zeile zu welcher *If*-Anweisung gehört.

Damit die Verschachtelung von *If*-Anweisungen nicht zu unübersichtlich wird, stellt VBA eine weitere Form der Verzweigung zur Verfügung, die *If-Then-Else*-Anweisung.

BSP_IFTHENELSEIF

```
Private Sub BSP_IFTHENELSEIF_Click()
   Dim X As Integer
   X = InputBox("Bitte eine ganze Zahl eingeben!", "EINGABE", "0")
        If X = 0 Then
                MsgBox "Zahl = 0"
        ElseIf X > 0 Then
                MsgBox "Zahl > 0"
        Else
                MsgBox "Zahl < 0"
        End If
End Sub
```

Bei dieser Form der Verzweigung wird zunächst auch die *If*-Bedingung überprüft. Falls diese nicht erfüllt ist, wird die erste *ElseIf*-Bedingung überprüft und falls sie erfüllt ist, deren Anweisungsblock ausgeführt. Falls sie nicht erfüllt ist, wird die nächste *ElseIf*-Bedingung überprüft usw. Wenn auch die letzte *ElseIf*-Bedingung nicht erfüllt ist, werden die Anweisungen des *Else*-Zweiges ausgeführt. Sobald eine *ElseIf*-Bedingung gefunden wurde, die erfüllt ist, werden die anderen nicht mehr überprüft.

Hinweis:

Die *If-Then-ElseIf*-Verzweigung muss keinen *Else*-Zweig haben. Dieser ist optional. Wird er weggelassen, wird im Extremfall kein Zweig der Verzweigung überprüft.

Select-Anweisung

Die *Select*-Anweisung ist eine Mehrfachauswahl, mit deren Hilfe aus einer Vielzahl von Möglichkeiten ausgewählt werden kann.

BSP_SELECT

```
Private Sub BSP_SELECT_Click()
        Dim strBuchstabe As String
        strBuchstabe = Inputbox("Bitte Buchstaben eingeben!", "EINGABE", "A")
        Select Case strBuchstabe
        Case Is = "A"
                MsgBox "Sie haben die Option A gewählt."
        Case Is = "B"
                MsgBox "Sie haben die Option B gewählt."
        Case Is = "C"
                MsgBox "Sie haben die Option C gewählt."
        Case Else
                MsgBox "Sie haben eine andere Option als A, B oder C gewählt!"
        End Select
End Sub
```

In diesem Beispiel stellt die Variable *strBuchstabe* den Wert dar, der überprüft wird. Jede *Case*-Zeile stellt eine Alternative dar, die überprüft wird. Wird die *Case Else* erreicht, ohne dass eine zutreffende Alternative gefunden wurde, so werden die zum *Else*-Zweig gehörenden Anweisungen ausgeführt.

Sie können auch für die einzelnen *Case*-Zweige bestimmen, dass mehrere alternative Werte für die Variable in Frage kommen. Dazu führen Sie die einzelnen Werte einfach der Reihe nach auf. Im Beispiel ist dargestellt, wie die Eingabe des Buchstabens von der Groß- und Kleinschreibung unabhängig gemacht wird.

BSP_SELECT2

```
Private Sub BSP_SELECT2_Click()
        Dim strBuchstabe As String
        strBuchstabe = Inputbox("Bitte Buchstaben eingeben! ", "EINGABE", "A")
        Select Case strBuchstabe
        Case Is = "A", "a"
                MsgBox "Sie haben die Option A gewählt."
        Case Is = "B","b"
                MsgBox "Sie haben die Option B gewählt."
        Case Is = "C","c"
                MsgBox "Sie haben die Option C gewählt."
        Case Else
                MsgBox "Sie haben eine andere Option als A, B oder C gewählt!"
        End Select
End Sub
```

Bestehen die alternativen Werte für die Variable, die überprüft wird, aus längeren Zeichenketten, müssten Sie hier alle möglichen Kombinationen aus großen und kleinen Buchstaben aufführen, um sicherzustellen, dass auch bei Tippfehlern die richtige Alternative gewählt wird. Dies kann natürlich eine lange Liste geben. Das folgende Beispiel zeigt, wie Sie solche Probleme durch Manipulation der Vergleichsvariablen ausschalten.

BSP_SELECT3

```
Private Sub BSP_SELECT3_Click()
        Dim strBuchstabe As String
        strBuchstabe = Inputbox("Bitte Buchstaben eingeben! ", "EINGABE", "A")
        Select Case Ucase(strBuchstabe)
        Case Is = "A"
                MsgBox "Sie haben die Option A gewählt."
        Case Is = "B"
                MsgBox "Sie haben die Option B gewählt."
        Case Is = "C"
                MsgBox "Sie haben die Option C gewählt."
        Case Else
                MsgBox "Sie haben eine andere Option als A, B oder C gewählt!"
        End Select
End Sub
```

Die Funktion *UCase* wandelt den als Parameter übergebenen String, hier *strBuchstabe*, in Großbuchstaben um. Sie brauchen also nur die Großbuchstaben abzufragen, um alle Möglichkeiten zu beachten. Entsprechendes können Sie natürlich auch mit der Funktion *LCase* erreichen, die alle Buchstaben in kleine verwandelt. In diesem Fall müssen Sie natürlich die Kleinbuchstaben in den *Case*-Zweigen abfragen.

Hinweis:

Bei der Formulierung der *Case*-Bedingungen müssen Sie auch nicht für alle Bedingungen den gleichen Vergleichsoperator verwenden. Sie können durchaus mischen. Achten Sie jedoch beim Formulieren der Bedingungen darauf, dass immer nur eine Bedingung zutrifft. VBA reagiert zwar nicht mit einer Fehlermeldung darauf, aber es kann zu anderen Ergebnissen führen, als Sie erwarten würden. Treffen mehrere Alternativen zu, wird nur der erste *Case*-Zweig ausgeführt, die anderen werden gar nicht mehr ausgeführt.

2. Schleifen

Bei den Schleifen unterscheidet man abweisende Schleifen, nicht abweisende Schleifen , Zählschleifen und Endlosschleifen. VBA bietet so viele verschiedene Schleifenvarianten wie kaum eine andere Programmiersprache. Im Prinzip käme man jedoch mit einer abweisenden und einer nicht abweisenden Schleife aus.
Alle Schleifen außer Endlosschleifen können vorzeitig verlassen werden. Bei abweisenden Schleifen heißt vorzeitig, obwohl die Eintrittsbedingung noch erfüllt ist, und bei nicht abweisenden bedeutet vorzeitig, dass die Austrittsbedingung noch nicht erfüllt ist. Bei Endlosschleifen ist ein vorzeitiges Verlassen aus definitorischen und praktischen Gründen nicht möglich. Eine Schleife, die keine Ende hat, kann auch keinen Zeitpunkt haben, der vor dem Ende liegt, um *vorzeitig* verlassen zu werden. Außerdem wäre die Schleife keine Endlosschleife mehr, wenn sie eine Anweisung enthielte, die in irgendeiner Situation dazu führt, dass die Schleife beendet wird.
Schleifen, die durch das Schlüsselwort *Do* eingeleitet werden, werden durch die Anweisung *Exit Do* verlassen, Schleifen, die mit *For* begonnen werden, können Sie mit *Exit For* verlassen.

Abweisende Schleifen

Abweisende Schleifen bestehen aus dem Schleifenrumpf, der die auszuführenden Anweisungen enthält und einer Eintrittsbedingung, die bestimmt, ob der Schleifenrumpf ausgeführt wird.

Hinweis:
Ist die Eintrittsbedingung von Anfang an nicht erfüllt, wird die Schleife gar nicht ausgeführt!

VBA kennt als abweisende Schleife die

- *Do-While-Loop*-Schleife und die
- *Do-Until-Loop*-Schleife.

Beide Schleifen können für die gleichen Probleme eingesetzt werden, dabei kommt es nur auf die konkrete Formulierung der Eintrittsbedingung an.

Die folgenden Beispiele zeigen, wie mit Schleifen die Summe aller Zahlen von 1 bis 10 berechnet werden kann. Dieses Problem wird einmal mit der *Do-While-Loop*-Schleifen und einmal mit der *Do-Until-Loop*-Schleife gelöst.

BSP_DoUntil

```
Private Sub BSP_DoUntil_Click()
        Dim X As Integer
        Dim Summe As Integer
        X = 0
        Do Until X = 10
                X = X + 1
                Summe = Summe + X
        Loop
        MsgBox "Die Summe beträgt :  " & Summe
End Sub
```

In diesem Beispiel wird der Schleifenrumpf ausgeführt, bis X den Wert 10 hat. Der Schleifenrumpf besteht aus der Anweisung X=X+1, die dafür sorgt, dass die Eintrittsbedingung irgendwann nicht mehr erfüllt ist, da sie in jedem Schleifendurchlauf X um 1 erhöht. Außerdem sorgt die Anweisung S*umme = Summe + 1* im Schleifenrumpf dafür, dass in der Variablen *Summe* die Summe aller X-Werte berechnet und gespeichert wird.

Hinweis:

An welcher Stelle die Anweisung X=X+1 im Schleifenrumpf steht, ist unerheblich, eine solche Anweisung muss nur vorhanden sein, da sonst aus einer Schleife eine Endlosschleife würde.

Natürlich ist dies nicht die einzige Lösungsmöglichkeit für das Problem, sowohl die Reihenfolge der Anweisungen als auch die Eintrittsbedingung könnte anders gestaltet werden. Eine andere Möglichkeit könnte so aussehen.

BSP_DoUntil2

```
Private Sub BSP_DoUntil2_Click()
        Dim X As Integer
        Dim Summe As Integer
        X = 1
        Do Until X > 10  'oder X = 11
                Summe = Summe + X
                X = X + 1
        Loop
        MsgBox "Die Summe beträgt :  " & Summe
End Sub
```

Beachten Sie hier, dass Sie auch den Anfangswert für die Variable X ändern müssen, wenn die Anweisung X=X+1 erst nach der ersten Summierung steht. In diesem Beispiel würde dies zwar nicht zu einem fehler führen, wäre aber ein verschwenderischer Schleifendurchlauf, da sonst der Schleifendurchlauf die Aufgabe Summe=0+0 lösen würde. Wenn Sie komplizierte Anweisungen im Schleifenrumpf ausführen, kann eine überflüssige Runde erhebliche Zeit kosten.

Hinweis:

Außerdem müssen Sie beachten, dass die Eintrittsbedingung nun nicht mehr X=10, sondern X=11 ist. Die falsche Formulierung der Abbruchbedingung bzw. Eintrittsbedingung ist die Fehlerquelle Nr. 1 bei Schleifen!!!

Die *Do-While-Loop*-Schleife führt den Schleifenrumpf aus, *solange* die formulierte Bedingung erfüllt ist. Die Bedingung muss also das genaue Gegenteil der Bedingung der *Do-Until-Loop*-Schleife sein. Mit der *Do-While-Loop*-Schleife sähe die Lösung des Problems wie folgt aus:

BSP_DoWhile

```
Private Sub BSP_DoWhile_Click()
        Dim X As Integer
        Dim Summe As Integer
        X = 1
        Summe = 0
        Do While X < 11
                Summe = Summe + X
                X = X + 1
        Loop
        MsgBox "Die Summe beträgt :  " & Summe
End Sub
```

Aber auch hier gibt es natürlich alternative Lösungen!

Nicht abweisende Schleifen

Die nicht abweisenden Schleifen führen, im Gegenteil zu den abweisenden Schleifen, den Schleifenrumpf mindestens einmal aus! Sie haben eine Austrittsbedingung, d.h. eine Bedingung die nach Ausführung des Schleifenrumpfes überprüft, ob die Bedingung zum Verlassen der Schleife erfüllt ist.

VBA kennt als nicht abweisende Schleifen

- *Do-Loop-Until*
- *Do-Loop*-While

Auch ihre Verwendung soll anhand der Summe der Zahlen von 1 bis 10 gezeigt werden.

BSP_DoLoopUntil

```
Private Sub BSP_DoLoopUntil_Click()
        Dim X As Integer
        Dim Summe As Integer
        Summe = 0
        X = 1
        Do
                Summe = Summe + X
                X = X + 1
        Loop Until X = 11
        MsgBox "Die Summe beträgt :  " & Summe
End Sub
```

Hinweis:

Eine Fehlerquelle liegt hier in der Formulierung der Abbruchbedingung. Die im Beispiel gewählte funktioniert nur, weil durch die Wertzuweisung X=1 vor Eintritt in die Schleife sichergestellt ist, dass X den Wert 1 hat und so die Abbruchbedingung auf jeden Fall irgendwann erfüllt ist. Wäre der Anfangswert variabel, so könnte es passieren, dass z.B. X bei Eintritt in die Schleife den Wert 11 hat. Dann würde der Schleifenrumpf durchlaufen und die letzte Zeile X=X+1 wurde X auf 12 erhöhen, damit wäre die Austrittsbedingung X=11 verfehlt und die Schleife würde unendlich fortlaufen, weil durch das ständige Erhöhen von X um 1 nie mehr der Wert 11 erreicht werden könnte.

Besser wäre in einem solchen Fall die folgende Abbruchbedingung:

```
Private Sub BSP_DoLoopUntil2_Click()
        …
        Do
                …
                X = X + 1
        Loop Until X > 10
        ...
End Sub
```

BSP_DoLoopUntil2

```
Private Sub BSP_DoLoopUntil2_Click()
        Dim X As Integer
        Dim Summe As Integer
        Summe = 0
        X = 1
        Do
                Summe = Summe + X
                X = X + 1
        Loop Until X > 10
        MsgBox "Die Summe beträgt :  " & Summe
End Sub
```

Die *Do-Loop-While*-Schleife führt den Schleifenrumpf wie die *Do-While-Loop*-Schleife aus, solange die Bedingung erfüllt ist. Allerdings überprüft sie die Bedingung erst, nachdem der Schleifenrumpf bereits einmal durchlaufen wurde. Die Summenberechnung sieht mit dieser Schleife wie folgt aus:

BSP_DoLoopWhile

```
Private Sub BSP_DoLoopWhile_Click()
        Dim X As Integer
        Dim Summe As Integer
        Summe = 0
        X = 1
        Do
                Summe = Summe + X
                X = X + 1
        Loop While X <= 10 ‘oder X < 11
        MsgBox "Die Summe beträgt :  " & Summe
End Sub
```

Auch hier kommt es auf den Anfangswert der Position der Anweisung X=X+1 und der formulierten Abbruchbedingung an, ob die Schleife zum richtigen Ergebnis führt und die Abbruchbedingung irgendwann erfüllt ist.

Zählschleifen

Zählschleifen werden verwendet, wenn zum Zeitpunkt der Programmierung bereits feststeht, wie oft die Schleife durchlaufen werden muss, wie bei unserer Summenberechnung. Andere Probleme lassen sich hierdurch nur mit umständlichen zusätzlichen Austrittsbedingungen formulieren.
Bei der *For-To-Next*-Schleife wird explizit angegeben, wie häufig die Schleife durchlaufen wird. Dazu wird eine Zählvariable verwendet, die meistens vom Typ *Byte*, *Integer* oder *Long* ist. Allerdings lässt VBA auch andere Datentypen für die Zählvariable zu. Es müssen nur numerische Datentypen sein.

Die Summe der Zahlen von 1 bis 10 wurde mit der *For-To-Next*-Schleife wie folgt bezeichnet werden:

BSP_ForToNext

```
Private Sub BSP_ForToNext_Click()
        Dim X As Integer
        Dim Summe As Integer
        Summe = 0
        For X = 1 To 10 Step 1
                Summe = Summe + X
        Next X
        MsgBox "Die Summe beträgt :  " & Summe
End Sub
```

Mit For X=1 wird der Variablen X der Anfangswert 1 zugewiesen. Dadurch ersparen Sie sich bei einer solchen Aufgabenstellung schon die explizite Zuweisung des Anfangswertes X=1 vor Eintritt in die Schleife. Dies erspart schon eine Zeile. Mit *To 10* wird der Endwert und damit die Abbruchbedingung für die Schleife festgelegt. Die Angabe *Step 1* ist optional und gibt an, wie groß die Schrittweite der Erhöhung der Variablen X sein soll. Standardmäßig ist *Step=1* und braucht in diesen Fällen nicht angegeben zu werden.

Der Schleifenrumpf besteht in dieser Schleife ausschließlich aus der Formel zur Summenbildung, da die Erhöhung der Variablen X durch die Schleife erfolgt und nicht explizit angegeben werden muss. *Next* schließt die Schleife ab. Die Angabe der Zählvariablen hinter *Next* ist seit dieser Version von VBA optional.

Zählschleifen sind in dieser Standardvariante wesentlich weniger fehleranfällig, da hier weniger Fehlerquellen vorhanden sind als bei den *Do-Loop*-Schleifen. Die Abbruchbedingung wird durch den maximalen Wert für die Zählvariable vorgegeben und der Programmierer braucht hier auch nicht selber dafür zu sorgen, dass die Abbruchbedingung irgendwann eintritt.

Hinweis:

Allerdings können auch hier Fehler gemacht werden, wenn optionale Angaben wie *Step* falsch gesetzt werden. Sie können nämlich auch einen negativen Wert als *Step*-Angabe verwenden. Dies führt dann dazu, dass die Zählvariable nicht erhöht, sondern erniedrigt wird. In diesem Fall müssen natürlich auch Anfangs- und Endwert entsprechend vertauscht werden, da sonst die Abbruchbedingung nie erfüllt wird. Allerdings erkennt VBA dies und führt dann den Schleifenrumpf gar nicht aus, was zwar zu einem falschen Ergebnis führt aber wenigstens nicht mehr zu einer Endlosschleife. Das Problem ist nur, dass VBA keinen Fehler meldet, die Endlosschleife vorher aber schnell erkannt und der Fehler behoben werden könnte. Dies ist bei einem falschen Ergebnis natürlich problematischer.
Richtig müsste die Schleife zur Summierung folgendermaßen aussehen, wenn sie rückwärts durchlaufen werden soll:

BSP_ForToNext2

```
Private Sub BSP_ForToNext2_Click()
        Dim X As Integer
        Dim Summe As Integer
        Summe = 0
        For X = 10 To 1 Step -1
                Summe = Summe + X
        Next X
        MsgBox "Die Summe beträgt :  " & Summe
End Sub
```

VBA bietet aber noch eine zweite Zählschleife. Bei ihr wird nur implizit eine Zählvariable definiert. Sie dient dazu, ein Listenobjekt, eine Aufzählung oder ein Array zu durchlaufen. Dabei ermittelt VBA selbst Anfangs- oder Endwert. Die Schleife endet automatisch, wenn das letzte Element der Liste abgearbeitet wurde.
Visual Basic, einschließlich VBA, ist eine der wenigen Programmiersprachen, die eine solche Schleife anbietet. Sie werden merken, wie nützlich sie insbesondere bei der Bearbeitung von Objekten sein kann. Trotzdem sollte die folgende Zählschleife mit Vorsicht zu genießen sein!

Das folgende Beispiel zeigt, wie mithilfe der *For-Each-Next*-Schleife eine Liste der in der aktuellen Arbeitsmappe vorhandenen Blätter ausgegeben werden kann.

BSP_ForEachNext

```
Private Sub BSP_ForEachNext_Click()
    Dim Blatt As Object
    For Each Blatt In ActiveWorkbook.Sheets
        Debug.Print Blatt.Name
    Next Blatt
End Sub
```

Dieser Programmcode kann bei Bedarf in *Excel* getestet werden.

Dazu wird zunächst eine Objektvariable definiert. Objektvariablen sind Variablen vom Datentyp *Object*. Diese Variable wird von der Schleife benötigt, um die Liste *Sheets*, die alle Blätter der aktuellen Arbeitsmappe enthält, zu durchlaufen. Dabei weist sie nacheinander der Objektvariablen die Blätter der Liste *Sheets* zu und gibt im Schleifenrumpf den Namen des Blattes aus, auf das die Objektvariable gerade verweist.

Endlosschleifen

Jede *Do-Loop*-Schleife können Sie zu einer Endlosschleife machen, indem Sie eine Eintrittsbedingung definieren, die immer erfüllt ist oder eine Austrittsbedingung, die nie erfüllt ist.

Hinweis:

Vor dem Test des folgenden Beispiels sollten Sie vorsichtshalber alle geöffneten Dateien speichern, falls beim Abbruch der Endlosschleife Ihre Anwendung abstürzen sollte. Endlosschleifen können Sie mit ESC abbrechen bzw. mit mehrmaligen Drücken der ESC-Taste.

Endlosschleifen werden in VBA mit *Do-Loop* formuliert. Dabei wird weder eine Eintrittsbedingung noch eine Austrittsbedingung angegeben. Das folgende Beispiel gibt einfach eine fortlaufende Nummer im Direktfenster aus, bis die Schleife mit ESC abgebrochen wird.

BSP_DoLoop

```
Private Sub BSP_DoLoop_Click()
    Dim X As Long
    Do
        X = X + 1
        Debug.Print X
    Loop
End Sub
```