AutoIt Programming Basics

Inhaltsverzeichnis

0) Hinweis

Alle enthaltenen Exkurse sind zur reinen Neugier da und schauen etwas bis viel über den Tellerrand hinaus. Sie sind enthalten, um ein besseres Verständnis zu ermöglichen.
Exkurse können auch übersprungen werden.
Exkurse müssen nicht unbedingt verstanden werden.

Alle Basics von AutoIt sind außerhalb der Exkurse beschrieben.

1) Ausgabe

Das erste, was man so gut wie in jeder Programmiersprache lernt ist die Ausgabe. Hier in AutoIt nutzen wir zur Ausgabe die Funktion *MsgBox()*. Diese Funktion verlangt folgende Parameter:

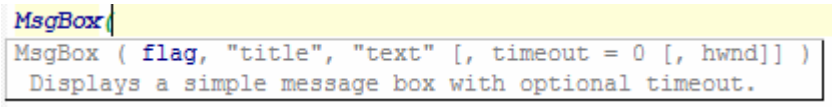


Abb 1. Parameterliste der Funktion MsgBox()

Die ersten Parameter 'flag', 'title' und 'text', müssen immer übergeben werden. Ab dem vierten Parameter 'timeout', ist es optional, ob man diesen und nachfolgende Parameter der Funktion mitgibt oder nicht. Optionale Parameter erkennt man daran, dass sie in eckigen Klammern stehen.

> Exkurs:
> Gibt man keinen Wert für optionale Parameter mit, so setzt der Compiler so genannte Standardwerte ein. In Abb 1 sieht man in der Parameterliste "... [, timeout = 0 ...]". Hier ist ein Hinweis für den Standardwert von 'timeout' zu sehen. Parameter, die man jedoch mitliefern muss, wie den Parameter 'flag', haben keinen Standardwert!

1.1) Die Funktion MsgBox()

MsgBox() erzeugt eine Ausgabe in einem Fenster. Hier ist zu beachten, dass dieses Ausgabefenster eine Form, einen Titel und einen Textbereich haben muss. Das verlangt AutoIt von jedem Programmierer, der diese Funktion nutzen möchte. Daher sind die folgenden 3 Parameter beim Funktionsaufruf Pflicht:

flag
flag beschreibt die Form des Fensters. Msgbox() hat eine Vielzahl von flags definiert, wie das Fenster letztlich aussieht und wie es sich verhält. Eine Liste der definierten flags findet man in der AutoIt-Hilfe. Hierzu klicken wir in unserem Programmcode auf die Funktion Msgbox() und drücken anschließend die Taste F1. Es öffnet sich die äußerst nützliche AutoIt-Hilfe. Sie springt direkt zum richtigen Eintrag, nämlich zur Erläuterung der Funktion Msgbox(). Siehe Abb 2. Scrollt man nach unten zu dem Punkt 'Remarks', sieht man die Liste aller definierten flags für den Parameter 'flag' für die Funktion MsgBox(). Sie Abb 3.

> Exkurs:
> Alle definierten flags in Abb 3 gelten nur für die Funktion MsgBox(). flags sind auf keine anderen Funktionen zu übertragen. Jede Funktion definiert ihre eigenen Parameter und ggf. auch flags.

Exkurs: Was ist ein "flag"?
In meinem Sprachgebrauch assoziiere ich das Wort flag mit "Beschreibung eines Zustands". Programmierer nutzen diesen Begriff häufig, um Fehler zu beschreiben.
Beispiel: Ein Programmierer schreibt die Funktion x(). Er lässt diese einmal durchlaufen und wenn alles glatt gelaufen ist, gibt die Funktion x() den Wert 0 zurück. Jedoch können bei einem Funktionsaufruf viele verschiedene Fehler passieren, gerade wenn der Nutzer des Programms aufgefordert wird, etwas einzugeben, und man möchte für jeden einzelnen Fehler, der auftreten kann, einen anderen Fehlerwert zurück geben. Wenn bspw. der Nutzer keinen Zugriff auf eine bestimmte Datei hat, geben wir den Wert 1 zurück, wenn seine Eingabe falsch war, geben wir 2 zurück etc. Anhand der Rückgabewerte können wir sehen, was falsch gelaufen ist. Hier setze ich wieder mit dem Begriff "Beschreibung eines Zustands" ein. Die Werte 0 (alles glatt gelaufen), 1 (Zugriff nicht gewährt) und 2 (falsche Eingabe), beschreiben einen Zustand. In der Funktion x() wird mit den Werten ein "Fehlerzustand" beschrieben. In der Funktion MsgBox() wird mit den Werten bzw. flags, der Zustand des Ausgabefensters beschrieben. Genauer gesagt wird mit dem flag in der Funktion MsgBox(), die Form des Ausgabefensters beschrieben.

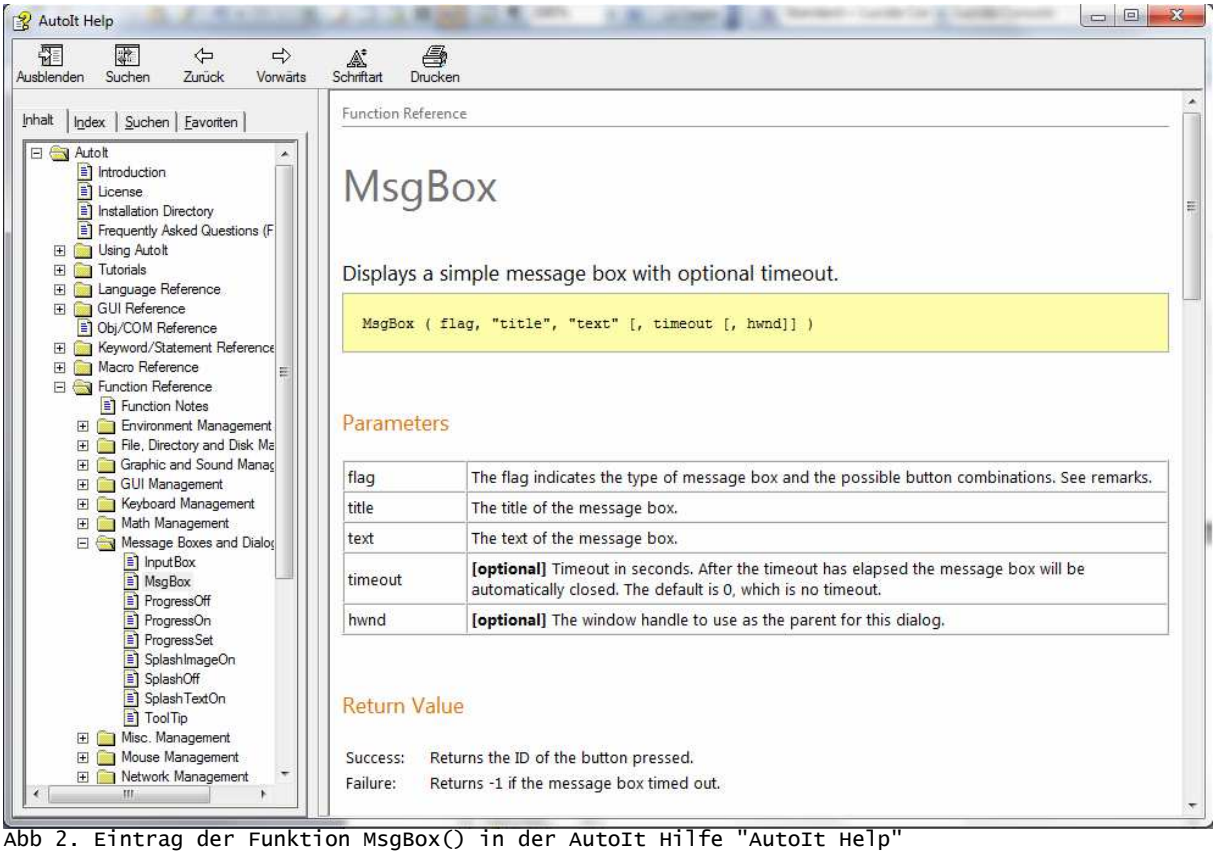
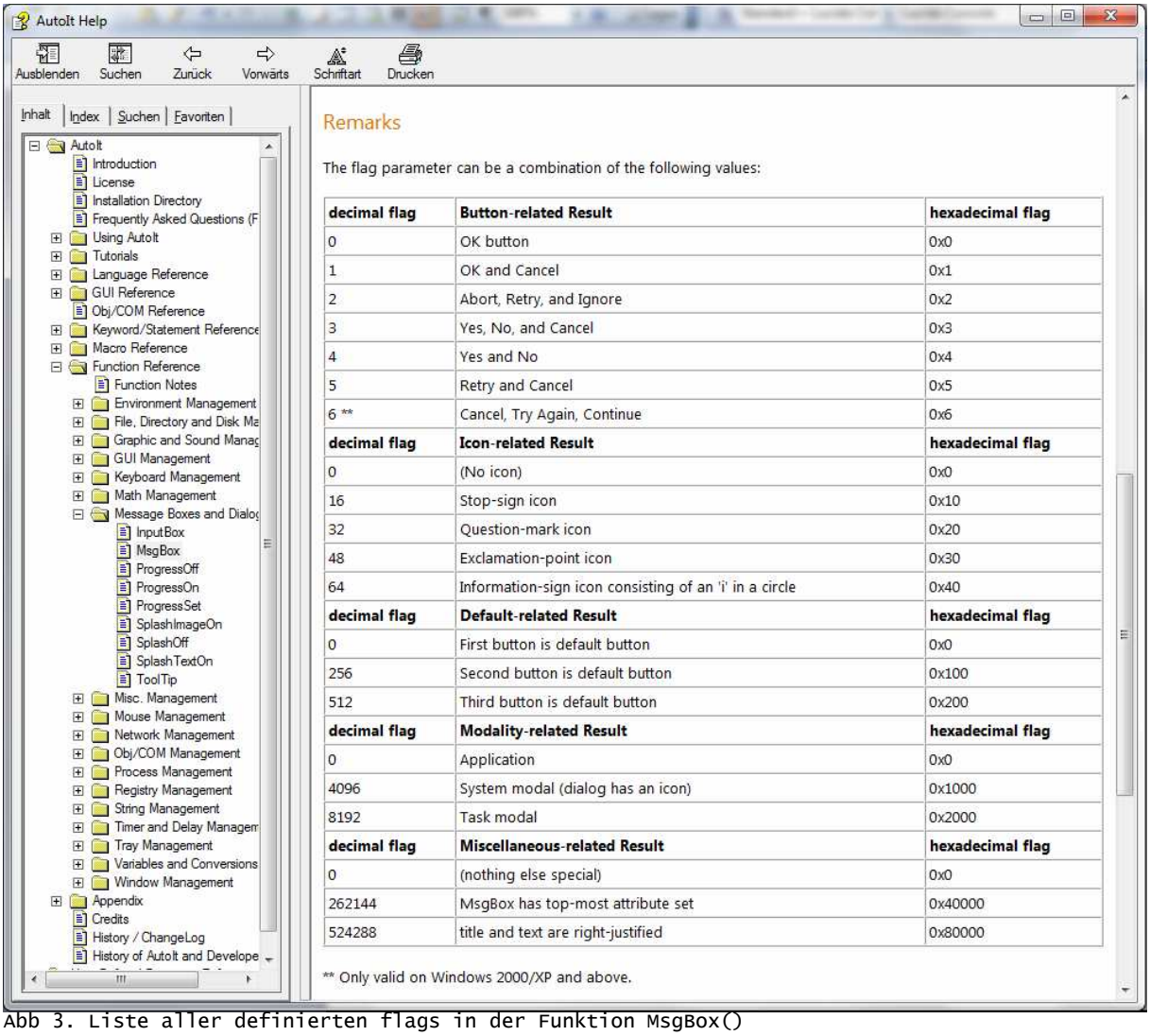


Abb 2. Eintrag der Funktion MsgBox() in der AutoIt Hilfe "AutoIt Help"



decimal flag	Button-related Result	hexadecimal flag
0	OK button	0x0
1	OK and Cancel	0x1
2	Abort, Retry, and Ignore	0x2
3	Yes, No, and Cancel	0x3
4	Yes and No	0x4
5	Retry and Cancel	0x5
6 **	Cancel, Try Again, Continue	0x6
decimal flag	**Icon-related Result**	**hexadecimal flag**
0	(No icon)	0x0
16	Stop-sign icon	0x10
32	Question-mark icon	0x20
48	Exclamation-point icon	0x30
64	Information-sign icon consisting of an 'i' in a circle	0x40
decimal flag	**Default-related Result**	**hexadecimal flag**
0	First button is default button	0x0
256	Second button is default button	0x100
512	Third button is default button	0x200
decimal flag	**Modality-related Result**	**hexadecimal flag**
0	Application	0x0
4096	System modal (dialog has an icon)	0x1000
8192	Task modal	0x2000
decimal flag	**Miscellaneous-related Result**	**hexadecimal flag**
0	(nothing else special)	0x0
262144	MsgBox has top-most attribute set	0x40000
524288	title and text are right-justified	0x80000

Abb 3. Liste aller definierten flags in der Funktion MsgBox()

Nun schauen wir uns die MsgBox() Funktion einmal in Aktion an. Folgende Programmcode Auszüge bewirken folgende Ausgaben:

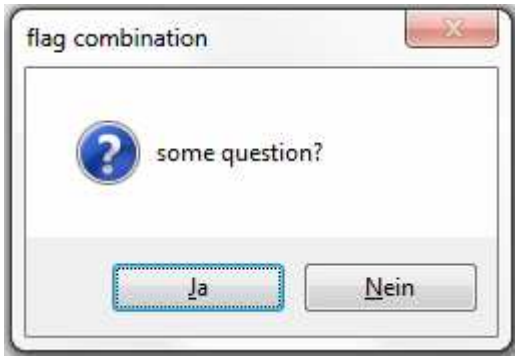
```
MsgBox(0, "First Program", "Hello World!")
```


```
MsgBox(4, "Second Program", "Still Hello World!")
```

Wie man sieht verändert 'flag' die Art des Ausgabefensters. Alle möglichen flags sind in Abb 3 abgebildet.

Es lassen sich auch flags kombinieren, wie das folgende Beispiel zeigt.

```
MsgBox(32 + 4, "flag combination", "some question?")
```



2) If, ElseIf und Else

Immer müssen wir, je nachdem wie sich das Programm verhält oder ob der Nutzer eine Eingabe macht, das Programm verzweigen lassen. Je nachdem welche Bedingung eingetreten ist, soll ein anderer Programmcode ausgeführt werden.

Die Syntax von einem einfachen If:

```
If condition Then
    statement
EndIf
```

Die Syntax von einem If mit Else-Verzweigung:

```
If condition Then
    statement1
Else
    statement2
EndIf
```

Die Syntax von If mit ElseIf- und Else-Verzweigung:

```
If condition1 Then
    statement1
ElseIf condition2 Then
    statement2
ElseIf condition3 Then
    statement3
Else
    statement4
EndIf
```

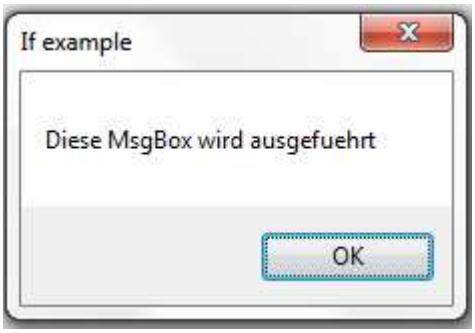
Da die Syntax nun abgeklärt ist, schauen wir uns If in Aktion an.

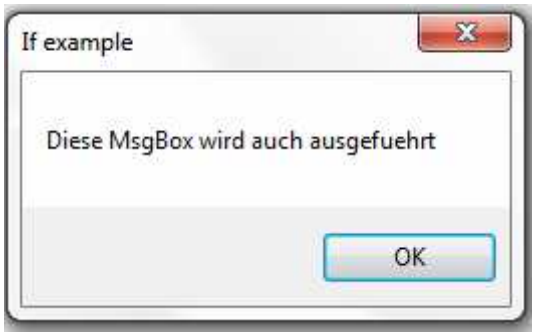
```
If 7 < 5 Then
    MsgBox(0, "If example", "Diese MsgBox wird nicht angezeigt")
ElseIf 2 < 5 Then
    MsgBox(0, "If example", "2 ist kleiner als 5")
Else
    MsgBox(0, "If example", "Falls keiner der Ifs zutrifft wird diese MsgBox angezeigt")
EndIf
```

Führt man dieses Code-Snippet aus erhält man folgende Ausgabe:

If überprüft die Bedingung, die zwischen 'If'/'ElseIf' und 'Then' steht. Ist die Bedingung war, wird der darunter stehende Ausdruck ausgeführt.
Wichtig bei einer If- mit ElseIf- und Else-Verzweigung ist, dass sobald eine Bedingung war ist und der Ausdruck hinter der Bedingung ausgeführt wurde, springt das Programm hinter 'EndIf' und überprüft keine weitere Bedingung.
Möchte man aber weiterhin noch mehrere Bedingungen überprüfen, so muss man mehrere einfache 'If'-Bedingungen ohne 'ElseIf'- und 'Else'-Bedingungen benutzen, was das folgende Beispiel zeigt.

```
If 7 < 5 Then
    MsgBox(0, "If example", "Diese MsgBox wird nicht angezeigt")
EndIf

If 6 < 5 Then
    MsgBox(0, "If example", "Diese MsgBox wird ebenfalls nicht angezeigt")
EndIf

If 5 < 5 Then
    MsgBox(0, "If example", "Diese MsgBox wird auch nicht angezeigt")
EndIf

If 4 < 5 Then
    MsgBox(0, "If example", "Diese MsgBox wird ausgefuehrt")
EndIf

If 3 < 5 Then
    MsgBox(0, "If example", "Diese MsgBox wird auch ausgefuehrt")
EndIf
```





3) Variablen

Variablen sind ein wichtiges Thema. Sie speichern alles ab, was du möchtest und du kannst die enthaltene Information jederzeit abrufen.
Um eine Variable zu deklarieren oder in anderen Worten, um sie zu erschaffen, ist folgender Programmcode erforderlich.

```
Dim $var
```

Das Schlüsselwort 'Dim' deklariert/erschafft die Variable. Der Ausdruck '$var' ist der Variablenname. Das Dollarzeichen muss vor jeder Variablen stehen, damit für den Compiler ersichtlich ist, dass es sich um eine Variable handelt.

Exkurs: Was ist deklarieren genau?
Deklarieren nennt sich der Vorgang, bei dem Speicherplatz für eine Variable reserviert wird. Zur Laufzeit des Programms, belegt das Programm eine gewisse Anzahl an Byte im Speicher/RAM. Variablen müssen ordentlich deklariert werden, damit der Compiler weiß wie viel Speicher er beim Betriebssystem anfragen muss.
Wenn man seinen Programmcode vom Compiler übersetzen lässt, weiß der Compiler zum Schluss welche Variablen alles existieren und wie viel Speicher einzelne Variablen benutzen. Der Speicherplatz wird jedoch schlussendlich vom Betriebssystem höchst persönlich zur Verfügung gestellt.

Exkurs: Was ist ein Compiler?
Jetzt wurde schon des öfteren das Wort Compiler in den Raum geschmissen, doch was ist das überhaupt?
Der Compiler übersetzt den Programmcode in, für die CPU ausführbare Maschineninstruktionen, auch Assembler genannt. Der Compiler ist immer die Instanz, die etwas an deinem Programmcode auszusetzen hat. Er gibt dir die Fehler in deinem Programmcode aus. Der Compiler kann keinen Programmcode übersetzen, den er nicht versteht. Die Übersetzung des Programmcodes erfolgt nach syntaktischen Regeln, also einer Form wie der Code auszusehen hat. Jede Programmiersprache verlangt eine andere Syntax, hat aber auch einen anderen Compiler/Übersetzer.
Ein Compiler ist intelligent. Wenn die Syntax nicht ganz ordentlich hingeschrieben ist, schätzt er manchmal ab, was wohl gemeint ist. In solchen Fällen gibt er oft Warnungen aus. Wenn Warnungen vorhanden sind, übersetzt der Compiler dennoch den Programmcode. Der Programmierer sollte aber auch die Warnungen des Compilers wahrnehmen, um eventuelle Programmabstürze zu verhindern.

So weist man einer Variablen einen Wert zu.

```
$var = 5
```

Variablen können ihren Wert jederzeit ändern, in was du auch möchtest.

```
$var = 5
$var = "string"
$var = 3.5
```

In AutoIt muss man nicht explizit Datentypen deklarieren. Der Compiler ermittelt den Datentyp immer automatisch. Daher ist es wichtig, seine Variablen bei der Deklaration auch gleich zu initialisieren, sprich der Variablen einen Wert zuzuweisen, damit der Compiler von Anfang an weiß, ob die Variable bspw. einen Text oder eine Zahl speichert.

So deklariert und initialisiert man Variablen in einem Schritt:

```
Dim $var = 0
```

Exkurs: Deklaration & Initialisierung
Es ist ein guter Programmierstil, wenn man sich angewöhnt, immer bei der Deklaration seine Variable auch zu initialisieren. Das hilft nicht nur den Datentyp von Anfang an zu bestimmten, sondern beugt auch Programmabstürzen vor. Wir stellen uns vor, wir deklarieren eine Variable, aber weisen ihr keinen Wert zu. Der Typ ist also unbekannt. Wenn wir Glück haben ermittelt der Compiler irgend einen Typ. Jetzt bauen wir die Variable in eine If-Abfrage ein oder vergleichen die Variable mit einer anderen. Das muss nicht zwingend zu einem Programmabsturz führen, aber führt zu einem unbekannten Verhalten des Programms. Wir wissen nicht mehr, was unser Programm macht und das darf nicht passieren. Der Compiler ist in AutoIt sehr freundlich. Er sagt nichts, wenn man seine Variablen nicht initialisiert.
Initialisiert man seine Variablen nicht, so steht ein Zufallswert in der Variablen. Das liegt daran, dass das Betriebssystem uns einen Speicherplatz zuweist, der höchst wahrscheinlich schon benutzt wurde. Wenn sich ein Programm beendet, dann setzt es nicht alles wieder auf null zurück, sondern gibt dem Betriebssystem einfach wieder den Speicher zurück, so wie das Programm ihn zuletzt benutzt hat.
Initialisiert man seine Variablen also alle zu Anfang an, so muss man sich weniger Gedanken machen.

Exkurs: Datentypen
Jede Variable hat einen Datentyp. Der Compiler von AutoIt bestimmt den Datentyp automatisch richtig.

Es gibt folgende Datentypen um ein paar aufzuzählen:

int (integer)	- für ganze Zahlen	- z.B. 3
float	- für Fließkomma Zahlen	- z.B. 3.7
char (character)	- für Zeichen	- z.B. a, #
string	- für Zeichnketten/Text	- z.B. Hallo

Über Datentypen müssen wir uns wenig Gedanken machen, da sich der Compiler damit beschäftigt. Was wir aber wissen müssen ist, dass bei einem Vergleich bspw. in einem If, wenn wir $var1 < $var2 vergleichen wollen, die beiden Variablen den gleichen Datentyp haben müssen, denn es bringt nichts wenn wir 3 < Hallo vergleichen, angenommen wir würden einen int mit einem string vergleichen. Damit kann der Compiler und wir selber auch nichts anfangen.

Möchte man von Anfang an eine Variable deklarieren, ihr aber erst später ihren Wert zuweisen, ist es üblich solche Variablen mit 0 zu initialisieren, wenn die Variable Zahlen speichert. Wenn sie einen Text oder nur ein einzelnes Zeichen später speichern soll, können sie mit einem Leerzeichen initialisiert werden. Das Leerzeichen ist relativ üblich, wenn man Text- oder Zeichenvariablen auf 0 bzw. leer initialisieren möchte.

Folgende Code-Snippets veranschaulichen dies.

0-Initialiserung einer ganzen Zahl (int):

```
Dim $var = 0
```

0-Initialisierung einer Fließkommazahl (float):

```
Dim $var = 0.0
```

Leer-Initialisierung eines Zeichens (char):

```
Dim $var = ' '
```

Leer-Initialisierung einer Zeichenkette (string):

```
Dim $var = " "
```

Es ist zu beachten, dass Zeichen mit einzelnen Hochkommata (') umschlossen werden und Zeichenketten mit doppelten Hochkommata ("). Schreibt man bspw. "a" in eine Variable, so schreibt man nur einen einzelnen Buchstaben in die Variable. Von der menschlichen Seite aus betrachtet ist es nur ein Zeichen, aber der Compiler unterscheidet " und '. "a" ist in dem Beispiel als Zeichenkette abgespeichert.

Exkurs:
Der Compiler muss Datentypen unterscheiden, weil die 7 bspw. als int, als float und als char Datentyp jeweils im Speicher unterschiedlich aussehen. Das liegt daran, dass Datentypen im Speicher unterschiedlich kodiert sind. Die jeweilige Kodierung ist darauf ausgelegt, gut mit ganzen Zahlen, Fließkommazahlen, Zeichen etc. umzugehen.
Wie schon gesagt, muss man bei logischen Ausdrücken, wie Vergleiche, darauf achten, dass die Datentypen übereinstimmen. Ansonsten kann es passieren, dass der Compiler Variablen falsch interpretiert. Es kann aber auch sein, dass der Compiler alles richtig macht und sogar in den richtigen Datentyp umwandelt. Da wir aber nicht hoffen wollen, dass der Compiler schon alles richtig machen wird, sollte man ein Auge darauf haben, was man mit was vergleicht. Für die Fehlersuche ist es nützlich, sich das im Hinterkopf zu behalten.

4) Eingabe

Hier wird erklärt, wie man für den Nutzer eine Eingabefläche schafft und anschließend darauf reagiert.

Die einfachste Eingabe schafft man mit der Funktion *InputBox()*.

Folgender Code-Snippet zeigt die Funktion:

```
InputBox(
InputBox ( "title", "prompt" [, "default" [, "password char"
 Displays an input box to ask the user to enter a string.
```

Anhand des Snippets sieht man, dass die Eingabe von "title" und "prompt" Pflicht sind. Alle weiteren Parameter sind optional. Das erkennt man daran, dass vor jedem nachfolgenden Parameter eine eckige Klammer steht.
Bei der Funktion ist die Liste der optionalen Parameter ein wenig länger und ist in dem Snippet nicht vollkommen aufgeführt.
Die optionalen Parameter von *InputBox()* werden in der Regel nicht benötigt.

Die Funktion *InputBox()* in Aktion:

Wir tragen folgende Zeichenketten (strings) als Parameter für die Funktion ein. Es ist wichtig, dass man Zeichenketten mit doppelten Hochkommata umschließt.

```
InputBox("First Input Box", "Please enter some data")
```

Die Ausgabe ist wie folgt.

Wir haben für den Parameter 'title' "First Input Box" übergeben. Man sieht, dass die Funktion diese Zeichenkette in den Titelbereich des Fensters eingetragen hat.
Wir haben für den Parameter 'prompt' "Please enter some data" übergeben. Man sieht, dass die Funktion diese Zeichenkette in den Fensterbereich eingetragen hat. Das, was man als 'prompt' übergibt, ist die Information für den Nutzer, was er eingeben soll.

4.1) Speichern einer Eingabe

So, wir haben unser Eingabefenster und der Nutzer trägt jetzt etwas in dem Eingabebereich ein und klickt auf 'OK'. Wie können wir nun die Eingabe intern verarbeiten?

Folgender Code-Snippet zeigt, wie man die Eingabe des Nutzers erstmal in einer Variablen speichert.

```
Dim $userInput = " "

$userInput = InputBox("First Input Box", "Please enter some data")
```

Wir deklarieren zu Beginn die Variable '$userInput', in der wir die Eingabe speichern wollen.
Uns gelingt es die Eingabe zu speichern, in dem wir "$userInput = InputBox(...)" schreiben.

Wie kommt aber die Zeichenkette jetzt in die Variable $userInput?

Hier noch mal unser Ausgangspunkt:

```
$userInput = InputBox("First Input Box", "Please enter some data")
```

Der Programmcode wird von links nach rechts ausgeführt.
Der Compiler (siehe Exkurs: Was ist ein Compiler?) liest beim Übersetzen des Programmcodes jedes einzelne Wort von links nach rechts ein. Soweit so gut.

Wir schauen uns also das erste Wort an: "$userInput". Der Compiler denkt sich: "Ok, eine Variable. Lesen wir das nächste Wort ein, um zu sehen, was damit passieren soll." Das nächste Wort ist: "=". Das Gleichheitszeichen ist der so genannte Zuweisungsoperator, dem wir schon öfters begegnet sind und (hoffentlich) automatisch richtig interpretiert haben. Seine Funktion ist genau die gleiche wie in der Mathematik. Er schreibt nämlich alles was auf der rechten Seite von ihm steht in die linke Seite. Die linke Seite ist immer eine Variable. Die rechte Seite kann theoretisch alles mögliche sein; eine Zahl, ein Text, ein Zeichen oder sogar eine Funktion wie in unserem Beispiel.
So, der Compiler weiß nun anhand dessen, was er eingelesen hat, nämlich bisher "$userInput =", dass er das, was er jetzt als nächstes einliest, in die Variable '$userInput' schreiben soll.
Wir schauen uns also das nächste Wort an: "InputBox("First Input Box", "Please enter some data")". Der Compiler merkt, dass es sich um eine Funktion handelt und führt diese aus. Erst nachdem die Funktion fertig durchgelaufen ist, macht der Compiler die Zuweisung. Die Funktion *InputBox()* ist so definiert, dass sie nach einem fertigen Durchlauf, einen Wert zurück gibt.

Gehen wir davon aus, der Nutzer würde das Wort "Hallo" in den Eingabebereich schreiben und auf 'OK' klicken.

Nach fertigem Durchlauf der Funktion kann man sich den Programmcode dann so vorstellen:

```
$userInput = "Hallo"
```

Die Funktion *InputBox()* gibt nämlich an die Stelle einen Wert zurück, an der sie aufgerufen wurde. Und sie gibt das, was der Nutzer eingegeben hat als Zeichenkette (string) zurück.

Somit wurde der Ausdruck

```
InputBox("First Input Box", "Please enter some data")
```

nach dem Funktionsaufruf zu

```
"Hallo"
```

ausgewertet.

Wir können uns das Ergebnis auch in einer Message Box mit folgendem Programmcode anzeigen lassen.

```
Dim $userInput = " "

$userInput = InputBox("First Input Box", "Please enter some data")

MsgBox(0, "input", $userInput)
```

Ausgaben:

4.2) Verarbeiten einer Eingabe

Wir können jetzt also die Eingabe des Nutzers in einer Variablen speichern und damit wie folgt weiter arbeiten.

In diesem Abschnitt werden wir ein kleines und sehr kurzes Spiel schreiben. Es soll den Namen tragen: "Welche Zahl suchen wir?".

Hier erstmal der Programmcode, die Erklärung folgt.

```
1
2   Dim $userinput = " "
3
4   $userinput = InputBox("Welche Zahl suchen wir?", "5 plus welche Zahl ergibt in der Summe 12?")
5
6   If ($userinput == 7) Then
7       MsgBox(0, "Richtig", "Du hast die gesuchte Zahl gefunden!")
8   Else
9       MsgBox(0, "Falsch", "Nein, die gesuchte Zahl ist eine andere.")
10  EndIf
11
12  Exit
13
```

Zu Beginn deklarieren wir ein Variable, in der wir die Eingabe des Nutzers speichern wollen, siehe Zeile 2 im Programmcode. Als nächstes lassen wir den Nutzer eine Eingabe machen, in der er die Lösung unseres kleinen Rätsels eingibt. Die Eingabe wird in der, in Zeile 2 deklarierten Variablen, gespeichert, siehe Zeile 4. Die Verarbeitung findet in dem If-Block von Zeile 6 bis 10 statt. Wir verarbeiten die Lösung, in dem wir die Eingabe mittels eines Ifs auf die richtige Lösung überprüfen, sprich wir prüfen den Ausdruck in dem If auf Gleichheit. Wurde die Lösung gefunden, sprich wurde der Ausdruck im If als wahr ausgewertet, teilen wir dem Nutzer mit, dass er die Lösung gefunden hat. Ist seine Lösung falsch, so wird der Else-Block vom If ausgeführt und wir teilen dem Nutzer mit, dass seine Lösung falsch ist. Hier ist zu

beachten, dass wenn man auf Gleichheit prüft, doppelte Gleichheitszeichen verwendet!!!
Das 'Exit' in Zeile 12 beendet dann nur noch das Programm, was in allen vorigen Code-Snippets zur Vereinfachung weggelassen wurde. Man sollte aber immer sein Programm ordentlich mit 'Exit' schließen, was wir jetzt auch in den folgenden Code-Snippets immer so handhaben werden.

Exkurs: Gleichheitsoperator (==) und Zuweisungsoperator (=)
Diese beiden Operatoren, haben jeweils unterschiedliche Bedeutungen im Programmcode. Mit '==' erzeugt man einen logischen Ausdruck, der überprüft, ob der Inhalt zweier Variablen identisch ist. Mit '=' macht man eine Zuweisung, also man schreibt den Wert der auf der rechten Seite steht in die Variable, die auf der linken Seite steht. Lange Rede kurzer Sinn, die Operatoren zu vertauschen oder nicht daran zu denken, dass es auch '==' gibt, kann fatale Folgen haben. Wir stellen uns vor wir überprüfen in einem If zwei Variablen auf Gleichheit. Wenn wir mit '=' vorhaben auf Gleichheit zu prüfen, dann ist der Ausdruck im If immer wahr, denn wir schreiben, dass was rechts steht links rein. Danach wird ausgewertet und natürlich sind beide Seiten identisch.
An dieser Stelle kommen wir wieder auf den Compiler zurück. Wie schon gesagt, ist er sehr freundlich. Sollte man unglücklicherweise statt '==' wirklich nur '=' schreiben, so deutet der Compiler den Programmierfehler automatisch richtig als '=='. Das ist eine Sache, die AutoIt so handhabt, aber keinesfalls in höheren Programmiersprachen vorzufinden ist.
Da aber generell jeglicher Programmcode, auch für andere Programmierer gut zu lesen sein soll, sollte man nicht die Freundlichkeit des Compilers ausnutzen und bei einer Prüfung auf Gleichheit immer '==' schreiben.

5) Schleifen

Als Schleifen bezeichnet man in der Programmierung, Abschnitte im Programm, die wiederholt ausgeführt werden. Diese Wiederholungen sind an Bedingungen gebunden, unter welchen Umständen sie ausgeführt und wie lange sie dann ausgeführt werden.
Es gibt mehrere Typen von Schleifen und die schauen wir uns jetzt alle einmal an.

5.1) While-Schleife

Das Grundgerüst einer While-Schleife sieht folgendermaßen aus:

```
While condition

    statement

WEnd
```

Die While-Schleife fängt mit dem Schlüsselwort 'While' an gefolgt von einer Bedingung (condition). Nur wenn diese Bedingung wahr ist, wird die Schleife ausgeführt. Ist sie nicht wahr, so macht das Programm hinter dem Schlüsselwort 'WEnd', was für 'While End' steht, weiter. 'WEnd' gibt bei der While-Schleife an bis wohin alles wiederholt werden soll, wenn dann auch die Bedingung wahr ist. 'statement' steht hier für alle Programmzeilen, die wir wiederholt haben möchten.

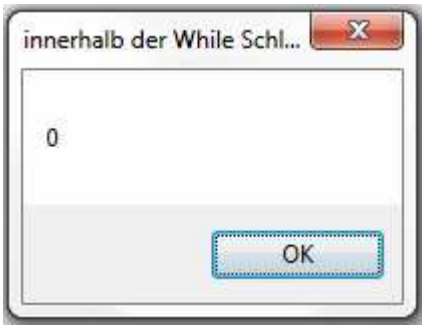
Folgender Programmcode zeigt die While-Schleife in Aktion.

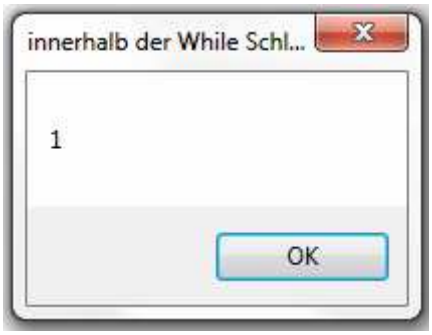
```
1
2   Dim $counter = 0
3
4   While $counter < 5
5
6       MsgBox(0, "innerhalb der While Schleife", $counter)
7
8       $counter = $counter + 1
9
10  WEnd
11
12  MsgBox(0, "außerhalb der While Schleife", "Die While Schleife wird nicht mehr ausgeführt")
13
14  Exit
15
```

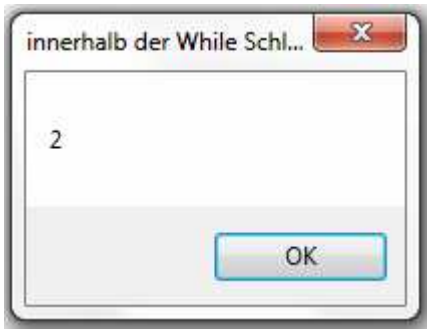
Wenn wir einen groben Blick auf den Teil mit der While-Schleife im Programmcode werfen, dann stellen wir schnell fest, dass die Bedingung dieser While-Schleife "$counter < 5" ist. Wenn dieser Ausdruck wahr ist, wird die Schleife ausgeführt. Und solange innerhalb dieser Schleife der Ausdruck immer noch wahr ist, wird die Schleife weiter ausgeführt. Das heißt für uns, dass wir als Programmierer immer darauf achten müssen, dass die Schleife auch irgendwann mal aufhört. Denn ein Programm, das nicht mehr aus seiner Routine rauskommen kann, muss wohl oder übel vom Task Manager beendet werden und das wollen wir nicht.
Um also eine Endlosschleife zu verhindern, sollte man am besten darauf achten, dass sich der Wert der Variable, die wir abprüfen, innerhalb der Schleife auch verändert. Das ist nämlich das, was im obigen Beispiel gemacht wurde. Unsere Bedingung heißt "$counter < 5", also die Schleife wird dann und solange ausgeführt, solange die Variable $counter einen Wert besitzt, der kleiner als 5 ist. Somit ist die Abbruchbedingung, dass $counter den Wert 5 oder größer haben muss, damit die Schleife abgebrochen wird. Im Beispiel wird die Variable $counter durch den Ausdruck "$counter = $counter + 1" bei jedem Durchlauf der Schleife um 1 erhöht.

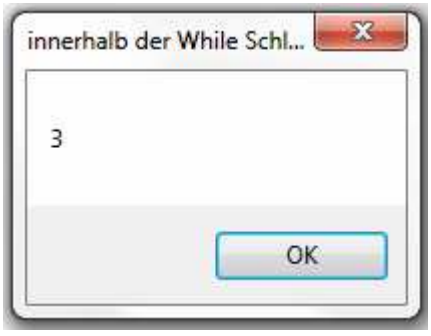
Der Sinn dieser While-Schleife ist, dass ein Programmabschnitt 5-mal wiederholt wird. Dazu setzen wir zu Beginn mit dem Programmcode "Dim $counter = 0" unsere Variable $counter auf 0. Die While-Schleife erkennt, dass 0 < 5 ein wahrer Ausdruck ist und führt die Schleife aus. In der Schleife haben wir eine Funktion *MsgBox()*, die uns zeigt welchen Wert $counter hat. Anschließend wird $counter erhöht. Unsere While-Schleife wird also ausgeführt, wenn unsere MsgBox den Wert 0, 1, 2, 3 und 4 zeigt. Wenn *MsgBox()* die Zahl 4 ausgibt, wird im nächsten Schritt die Variable $counter um 1 erhöht und hat dann den Wert 5. Da "$counter = $counter + 1" die letzte Programmzeile im Schleifeninneren ist, springt das Programm wieder an den Anfang der Schleife und überprüft die Bedingung "$counter < 5" auf Wahrheit. $counter ist an dieser Stelle 5. Da 5 < 5 ein falscher Ausdruck ist, wird die Schleife nicht noch einmal ausgeführt, das Programm springt hinter 'WEnd' und führt nachdem 4 ausgegeben wurde nun eine neue MsgBox aus. Es erscheint dann die Nachricht "Die While Schleife wird nicht mehr ausgeführt". Klickt man diese MsgBox weg, wird anschließend nur noch 'Exit' ausgeführt und das Programm beendet sich.

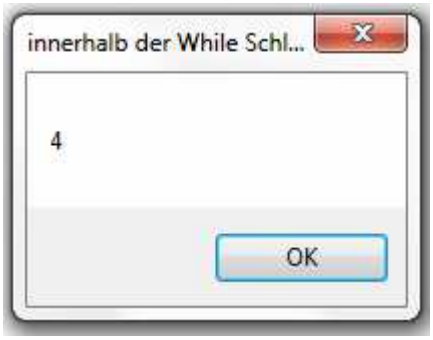
Ausgaben:

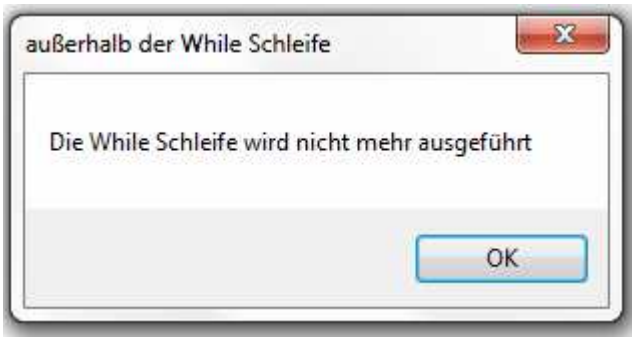












Exkurs: While-Schleife im kleinsten Detail (eine wage Vermutung)

Die While-Schleife macht nichts anderes als einen Programmteil zu wiederholen, aber wie schafft sie es wieder zum Anfang zu finden?

Seit je her werden Programme, die ausgeführt werden erst einmal in den Speicher geladen. Das heißt, dass jede einzelne Variable und jede einzelne Funktion von unserem Programm an einer bestimmten Adresse im Speicher steht, wo man sie finden kann. Auf unsere While-Schleife bezogen, steht ihr Anfang an einer festen Adresse im Speicher und die zu wiederholenden Befehle an den kommenden Adressen. Unser Programm führt jetzt unsere Schleife aus. Ist ein Durchlauf der Schleife beendet, so kommt sie wieder zu ihrem Anfang zurück, weil der Compiler noch einen aller letzten Befehl an das Ende der Schleife schreibt. Er schreibt noch einen Sprungbefehl hinten dran, der auf die Adresse verweist, wo die While-Schleife beginnt. Da der Compiler unser Programm übersetzt und er es letzten Endes besser kennt als wir selber, weiß er wo alles anfängt und wo alles aufhört.

5.1.1) Der ExitLoop Befehl

Der Befehl 'ExitLoop' beendet sofort die Ausführung einer Schleife und springt an ihr Ende. Im Falle der While-Schleife wird sofort hinter 'WEnd' gesprungen. Das funktioniert innerhalb einer While-Schleife, die wir eben besprochen haben, aber auch bei einer Do- und einer For-Schleife, die wir in den kommenden Abschnitten noch besprechen werden.

Selbst wenn wir eine Endlosschleife programmiert haben, kann man mit 'ExitLoop' heraus springen.

Folgender Code-Snippet zeigt eine Endlosschleife, aus der mit 'ExitLoop' heraus gesprungen wird, in Aktion.

```
1
2   Dim $counter = 0
3
4   While True
5
6       If $counter == 3 Then
7
8           ExitLoop
9
10      EndIf
11
12      MsgBox(0, "innerhalb der While Schleife", $counter)
13
14      $counter = $counter + 1
15
16  WEnd
17
18  MsgBox(0, "außerhalb der While Schleife", "Die While Schleife wird nicht mehr ausgeführt")
19
20  Exit
```

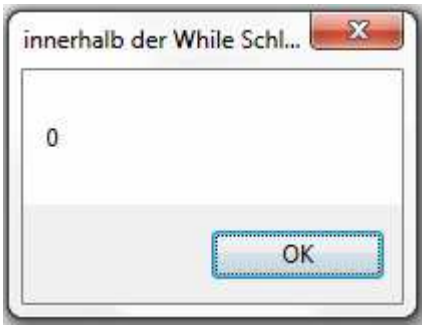
Das Programm ist eine Abwandlung aus Abschnitt "5.1) While-Schleife". Zeile 4 zeigt, dass man mittels "While True" eine Endlosschleife kreiert. Am Kopf der Schleife wird jetzt

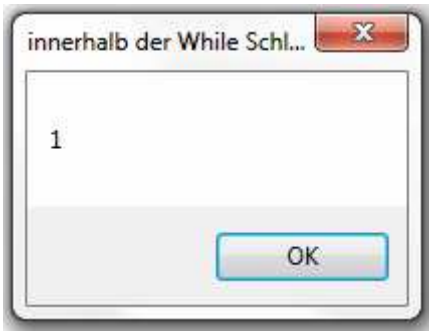
praktisch keine Bedingung mehr überprüft, denn die Bedingung für die While-Schleife ist immer wahr, weil wir 'True' als Bedingung geschrieben haben.

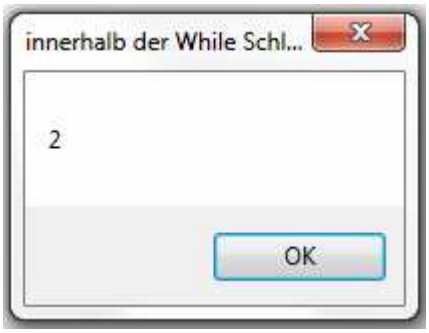
> Exkurs:
> Letzten Endes resultiert jeder logischer Ausdruck zu 'True'/wahr oder 'False'/falsch. Logische Ausdrücke können als Bedingungen geschrieben werden, die man dann für einen If-Ausdruck oder die Bedingung einer While-Schleife benutzen kann.
> Wenn man möchte, dass eine Bedingung immer wahr oder immer falsch ist, so kann man den logischen Ausdruck, wie $var1 < $var2, weglassen und dafür 'True' schreiben, wenn etwas immer ausgeführt werden soll oder 'False', wenn etwas niemals ausgeführt werden soll. Logisch ist es aber, wenn man etwas programmiert, das man eigentlich gar nicht ausgeführt haben möchte, einfach weglässt.

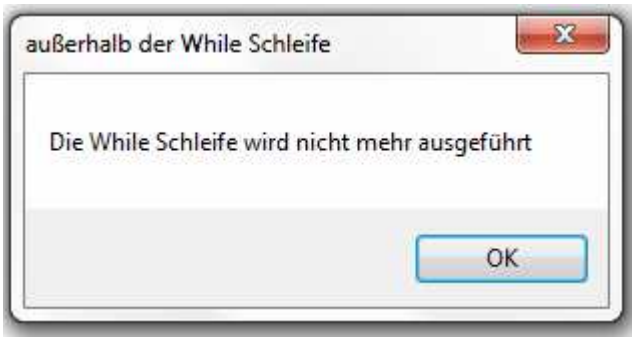
Wie auch in Abschnitt 5.1, haben wir eine Variable $counter, die zu Anfang auf 0 gesetzt und innerhalb der While-Schleife erhöht wird. Innerhalb der While-Schleife prüfen wir ab, ob $counter den Wert 3 hat, und wenn ja, dann wird der Befehl 'ExitLoop' ausgeführt und wir entkommen der Endlosschleife.

Ausgaben:









5.1.2) Der ContinueLoop Befehl

Der 'ContinueLoop' Befehl funktioniert im Prinzip wie der 'ExitLoop' Befehl, außer mit dem Unterschied, dass sofort an den Schleifenkopf gesprungen und die Bedingung der Schleife erneut überprüft wird. Falls die Bedingung wahr ist, wird die Schleife von neuem ausgeführt, anderen Falls wird hinter dem Schleifenende weiter gemacht.

Ein Code-Snippet wird für diesen Befehl nicht gezeigt. Es soll nur darauf hingewiesen werden, dass man auch an den Schleifenkopf springen kann, wenn man dies benötigt.

5.2) Do-Schleife

Das Grundgerüst einer Do-Schleife sieht wie folgt aus:

```
Do

    statement

Until condition
```

Eine Do-Schleife funktioniert im Prinzip ähnlich wie eine While-Schleife. Die While-Schleife prüft zu Beginn, ob ein Block Programmcode überhaupt ausgeführt werden soll. Die Do-Schleife hingegen prüft immer zum Schluss, wenn der Block Programmcode ausgeführt wurde, die Bedingung.
Die Do-Schleife wird also mindestens immer einmal ausgeführt. Die Logik der Do-Schleife ist allerdings in AutoIt umgekehrt wie bei der While-Schleife zu verstehen. Die While-Schleife wird solange ausgeführt, wie die Bedingung wahr ist. Die Do-Schleife allerdings wird solange ausgeführt bis die Bedingung wahr geworden ist. Das heißt die Do-Schleife wiederholt seinen Programmblock solange, wie der Ausdruck der unten abgeprüft wird, falsch ist, um das wirklich noch einmal zu betonen.
So eine Konstruktion verwendet man bspw. bei Passwortabfragen. Man wird mindestens einmal nach dem Passwort gefragt. Ist es richtig kommt man aus der Schleife wieder heraus, ist es falsch kann man die Abfrage beliebig oft wiederholen, je nachdem wie man das Innere der Do-Schleife programmiert.

Hier ein sehr simples Passwortabfrageprogramm:

```
1
2   Dim $password = "topsecret"
3   Dim $userInput = " "
4
5   Do
6
7       $userInput = InputBox("Passwort", "Geben Sie das Passwort ein", "", "*")
8
9   Until $userInput == $password
10
11  MsgBox(0, "Passwort", "Das Passwort war richtig!")
12
13  Exit
14
```

Zu Beginn deklarieren wir die Variablen, die wir brauchen. Einmal in Zeile 2 eine Variable, die unser Passwort beinhaltet, und einmal in Zeile 3 eine Variable, in der der Nutzer seine Eingabe macht. Die Do-Schleife fängt in Zeile 5 an und wie üblich bei Do-Schleifen wird am Anfang keine Bedingung geprüft. Es wird also sofort Zeile 7 ausgeführt und der Nutzer wird zu einer Eingabe aufgefordert. Zeile 7 beinhaltet eine *InputBox()*-Funktion, die wir bereits besprochen haben. Allerdings ist die Parameterliste dieser *InputBox()*-Funktion ein wenig länger als wir es kennen. Damit bei der Passwortabfrage statt des echten Passworts Sterne (*) angezeigt werden, muss man den 4. Parameter benutzen. In Anführungszeichen schreiben wir für diesen Parameter das Zeichen rein, das wir für jeden Buchstaben angezeigt haben möchten, den wir eingeben. *InputBox()* gibt allerdings an die Variable $userInput das zurück, was der Nutzer wirklich eingegeben hat.
Stimmt die Eingabe in $userInput mit unserem festgelegten Passwort, das in $password steht, überein, dann wird die Do-Schleife abgebrochen. Denn die Do-Schleife hört erst auf, sich zu wiederholen, wenn die Bedingung am Ende der Schleife wahr geworden ist.
Geben wir das richtig Passwort "topsecret" ohne Anführungszeichen ein, dann springt das Programm zu Zeile 11 und eine MsgBox erscheint. Klickt man diese weg, wird das Programm, durch das in Zeile 13 stehende Exit, beendet.

Exkurs: Eine Passwortabfrage mit 3 Versuchen

```
1
2   Dim $password = "topsecret"
3   Dim $userInput = " "
4   Dim $tries = 3
5
6
7   Do
8
9       $userInput = InputBox("Passwort", "Geben Sie das Passwort ein", "", "*")
10
11      $tries -= 1
12
13  Until ($userInput == $password) Or ($tries <= 0)
14
15
16  If $userInput == $password Then
17
18      MsgBox(0, "Passwort", "Geheime Dokumente geöffnet")
19
20  Else
21
22      MsgBox(0, "Passwort", "Sie haben alle 3 Versuche aufgebraucht. Das Programm beendet sich jetzt")
23
24  EndIf
25
26
27  Exit
```

Wenn der Nutzer 3 Fehlversuche gemacht hat, dann beendet sich das Programm. Anderen Falls ist der Zugriff auf die geheimen Dokumente gewährt.

5.3) For-Schleife

Das Grundgerüst der For-Schleife:

```
For start value To stop value Step amount

    statement

Next
```

For-Schleifen werden dazu benutzt, wenn man einen Codeblock (statement), eine BESTIMMTE Anzahl an Malen wiederholen möchte. Für diesen Zweck lässt man eine For-Schleife immer zählen. Nämlich von einem Anfangswert (start value), der in einer Variablen gespeichert werden muss, bis zu einem Endwert (stop value), der nicht unbedingt eine Variable sein muss. 'amount' gibt dann nur noch an in welchen Abständen hoch oder runter gezählt werden soll. Man kann bspw. von 0 bis 9 in 1er schritten zählen oder auch in 0.5er Schritten. Dazu ist der 'Step' da.

Folgende For-Schleife zählt von 0 bis 9 und wird somit 10-mal ausgeführt, weil wir von 0 anfangen zu zählen.

```
1
2   Dim $counter = 0
3
4   For $i = 0 To 9 Step 1
5
6       $counter += 1
7
8   Next
9
10  MsgBox(0, "For-Schleife", $counter)
11
12  Exit
```

Zeile 2 deklariert eine Variable $counter und weist ihr den Wert 0 zu. Mit $counter zählen wir im Inneren der For-Schleife mit, wie viel mal sie ausgeführt wird.

For $i = 0 To 9 Step 1

Sehr typisch in vielen Programmiersprachen ist es in der For-Schleife eine Variable zu deklarieren, die hoch gezählt wird. Wir waren es immer gewöhnt unsere Variablen vorher zu deklarieren als wir sie noch gar nicht gebraucht haben, aber die For-Schleife ist da die einzige Ausnahme, in der es wirklich ratsam ist die Deklaration und anschließende Initialisierung/Wertzuweisung im Kopf der For-Schleife zu machen, denn so ist der Programmcode für andere Programmierer besser zu lesen.

Ablauf einer For-Schleife

1) Ein Startwert wird festgelegt und in einer Variablen gespeichert. Oft nennt man aus Gewohnheit die Variable $i.

2) Es wird die Bedingung überprüft. For zählt nur soweit wie der Ausdruck hinter 'To' es sagt. In unserem Beispiel ist es 9. Wenn $i = 9 ist, wird die Schleife zum letzten mal ausgeführt. Es wird also immer EINSCHLIEßLICH bis zu dem Wert gezählt, den wir hinter 'To' festgelegt haben.

3) Das Innere der For-Schleife wird ausgeführt.

4) $i wird um den Step erhöht, den wir hinter Step festgelegt haben.

2) Schritt 2) tritt wieder in Kraft. Es wird die Bedingung überprüft. [...]

Das 'Next' in der For-Schleife ist so wie das 'WEnd' in einer While-Schleife zu verstehen. Die Wiederholung in der For-Schleife wird nur bis 'Next' ausgeführt.

6) Switch-Anweisungen

Das Grundgerüst einer Switch-Anweisung:

```
Switch $var

    Case 1
        statement1

    Case 2
        statement2

        .
        .
        .

    Case n
        statementn

    Case Else
        statementelse

EndSwitch
```

Eine Switch-Anweisung ist genau das gleiche wie eine If-Anweisung (vgl. Case 1) mit ElseIf-Anweisungen (vgl. Case 2-n) und einer Else-Anweisung (vgl. Case Else).

Der folgende Code-Snippet mit If realisiert, ist 1:1 mit Switch realisierbar, darauf nachfolgender Code-Snippet.

If Code-Snippet:

```
1
2   Dim $userInput = 0
3
4   $userInput = InputBox("Eingabe", "Geben Sie eine Zahl ein")
5
6   If $userInput == 1 Then
7
8       MsgBox(0, "", "Sie haben 1 eingegeben")
9
10  ElseIf $userInput == 2 Then
11
12      MsgBox(0, "", "Sie haben 2 eingegeben")
13
14  Else
15
16      MsgBox(64, "", "Ihre Zahl war nicht dabei")
17
18  EndIf
19
20  Exit
```

Switch Code-Snippet:

```
1
2   Dim $userInput = 0
3
4   $userInput = InputBox("Eingabe", "Geben Sie eine Zahl ein")
5
6   Switch $userInput
7
8       Case 1
9           MsgBox(0, "", "Sie haben 1 eingegeben")
10
11      Case 2
12          MsgBox(0, "", "Sie haben 2 eingegeben")
13
14      Case Else
15          MsgBox(64, "", "Ihre Zahl war nicht dabei")
16
17  EndSwitch
18
19  Exit
```

Da beide Code-Snippets genau das gleiche machen, ist es dem Programmierer überlassen, ob er lieber If oder lieber Switch benutzt. Switch wird allerdings oft If vorgezogen, wenn es zu viele Fälle gibt, die man abprüfen möchte. Denn mit Switch ist der Programmcode in solchen Fällen oft übersichtlicher.

Exkurs:

Switch kann man für alle möglichen Datentypen benutzen. Das folgende Beispiel zeigt Switch mit einer string/Zeichenkette Variablen.

```
1
2   Dim $userInput = " "
3
4   $userInput = InputBox("InputBox", "Eingabe für Switch")
5
6   Switch $userInput
7
8       Case "Hallo"
9           MsgBox(0, "Switch", "Ebenfalls Hallo")
10
11      Case "Wer bist du?"
12          MsgBox(0, "Switch", "Ich bin eine Switch Anweisung")
13
14      Case Else
15          MsgBox(16, "Switch", "Keiner der festgelegten Fälle ist eingetroffen")
16
17  EndSwitch
18
19  Exit
```

7) Arrays

Als Array betrachtet man bildlich eine große Variable, die mehrere Variablen enthält. Man kann also sagen, dass ein Array Variablen verwaltet.

Schauen wir uns einmal an wie man ein Array erschafft.

```
Dim $array[5]
```

Arrays werden genau so deklariert wie normale Variablen auch. Nur mit dem Unterschied, dass hinter dem Variablennamen eckige Klammern stehen und innerhalb dieser eckigen Klammer eine ganze Zahl steht. Diese Zahl, oder auch Index genannt, entspricht der Anzahl an Variablen, die das Array enthält. In unseren kommenden Beispielen werden wir den Index 5 nehmen. Der Index kann natürlich variiert werden wie man möchte, aber es muss eine ganze Zahl sein.

Wie greift man auf die Variablen eines Arrays zu?

Man greift mittels des Indexes darauf zu.
Das sieht folgendermaßen aus.

```
$array[0]    ; 1. Variable
$array[1]    ; 2. Variable
$array[2]    ; 3. Variable
$array[3]    ; 4. Variable
$array[4]    ; 5. Variable
```

So können wir in jede einzelne Variable, einen Wert schreiben, wie das folgende Beispiel zeigt.

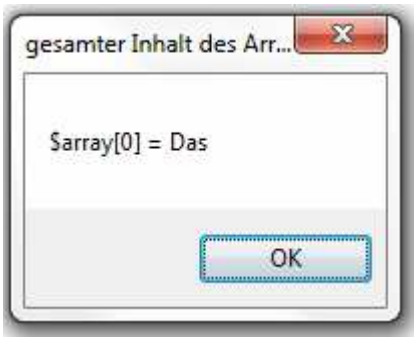
```
$array[0] = "Das"
$array[1] = "hier"
$array[2] = "ist"
$array[3] = "ein"
$array[4] = "Array"
```

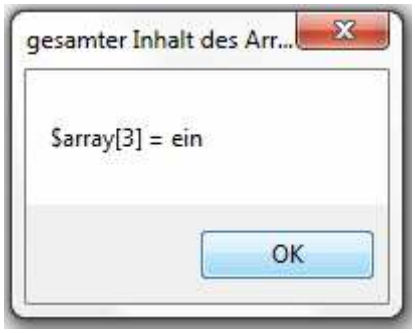
Den Inhalt eines Elementes zeigt man bspw. so an.

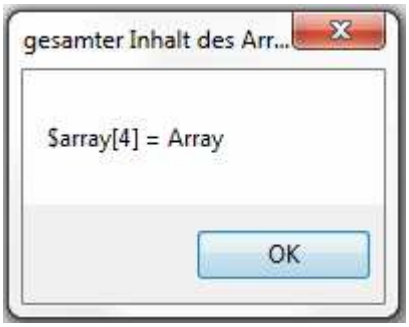
```
MsgBox(0, "", $array[0])
```

Um den gesamten Inhalt eines Arrays anzuzeigen benutzt man eine For-Schleife.

```
For $i = 0 To 4 Step 1

    MsgBox(0, "gesamter Inhalt des Arrays", "$array[" & $i & "] = " & $array[$i])

Next
```







Wir benutzen $i aus der For-Schleife und setzen es für unseren Index in unser Array ein. Dabei muss man nur darauf achten, was man für 'To' einsetzt. Will man unglücklicherweise mehr ausgeben als das Array an Elementen hat, kann das böse enden. Das Kaufmanns-Und '&' in der MsgBox dient dazu, Zeichenketten zu verbinden.

ACHTUNG!

Wir haben ein Array mit 5 Elementen deklariert.
Wenn man unglücklicherweise versucht mit $array[5] auf das 5. Element zuzugreifen, ist das ein Denkfehler.
Da Arrays immer mit dem Index 0 beginnen, muss man darauf achten, dass man mit $array[5] auf das 6. Element zugreift.
In unseren Beispielen gibt es nur 5 Elemente, also wäre der Zugriff auf $array[5] eine **Speicherverletzung!**
Speicherverletzung heißt in AutoIt **Programmabsturz!**
Der Compiler von AutoIt achtet beim Übersetzen nicht darauf, ob wir außerhalb des Bereichs unseres Arrays lesen oder schreiben wollen. Da AutoIt aber immer ein Sicherheitssystem beim Übersetzen einschleust, merkt er es aber zur Laufzeit, also wenn wir unser Programm ausführen. Manchmal gibt er uns eine Fehlermeldung aus und stürzt dann ab oder stürzt einfach sofort ab. Windows stürzt aber auf keinen Fall ab und läuft nach unseren Programmabstürzen fehlerfrei weiter.

Also...

nur Mut zum Programmabsturz. Testen wir einmal was passiert. Folgender Code bringt unser Programm zum Absturz.

```
1
2   Dim $array[5]
3
4   $array[0] = "Das"
5   $array[1] = "hier"
6   $array[2] = "ist"
7   $array[3] = "ein"
8   $array[4] = "Array"
9   $array[5] = "-Programmabsturz" <- Hier ist die Speicherverletzung
10
11  For $i = 0 To 4 Step 1
12
13      MsgBox(0, "gesamter Inhalt des Arrays", "$array[" & $i & "] = " & $array[$i])
14
15  Next
16
17  Exit
```

Mit 'Ctrl+F5' können wir unsere Syntax überprüfen lassen bzw. so fragen wir den Compiler, ob er bereit ist unseren Programmcode zu übersetzen. Machen wir das einmal und schauen, was der Compiler zu unserem fehlerhaften Code sagt.

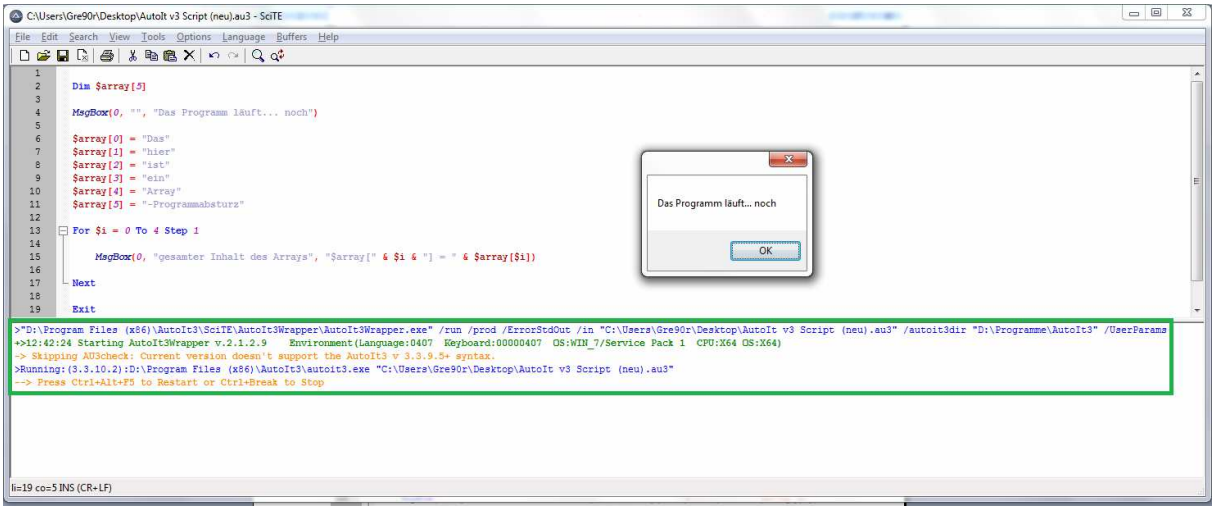
```
>"D:\Program Files (x86)\AutoIt3\SciTE\AutoIt3Wrapper\AutoIt3Wrapper.exe" /prod /AU3Check /in "C:\Users\Gre90r\Desktop\AutoIt v3 Script (neu).au3"
+>12:05:31 Starting AutoIt3Wrapper v.2.1.2.9    Environment(Language:0407  Keyboard:00000407  OS:WIN_7/Service Pack 1  CPU:X64 OS:X64)
>Running AU3Check (3.3.10.2)  from:D:\Program Files (x86)\AutoIt3
+>12:05:31 AU3Check ended.rc:0
>Exit code: 0    Time: 0.322
```

"AU3Check ended.rc:0" und keine rote Schrift sagt uns, dass der Compiler unseren Programmcode akzeptiert. (Es gibt einen Exkurs über Compiler auf Seite 7. An dieser Stelle wäre es ganz interessant, zu wissen was das ist.)

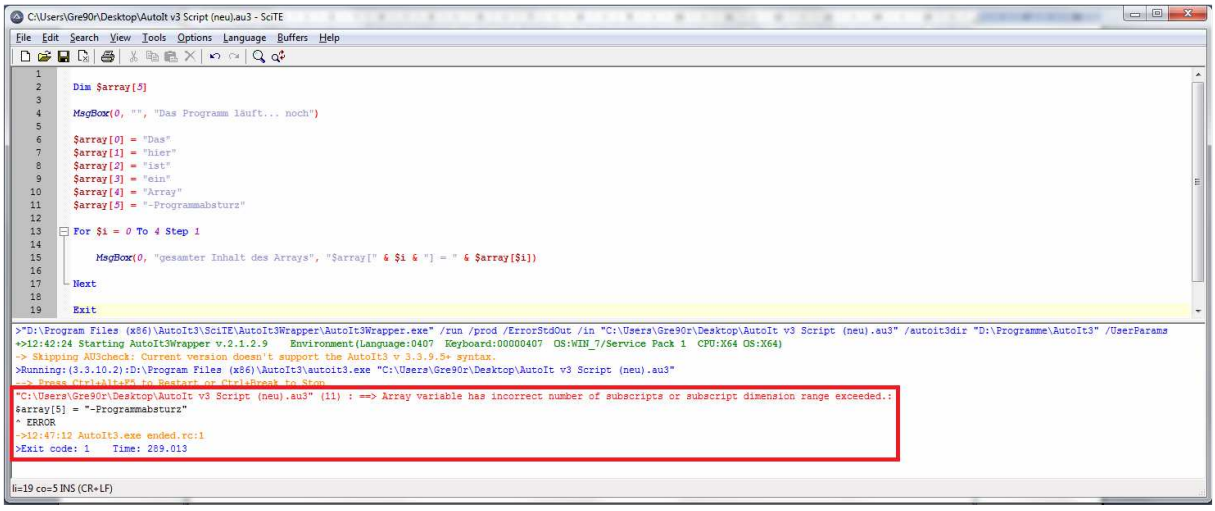
Führen wir den Programmcode aus, zeigt uns der Compiler folgendes.

```
>"D:\Program Files (x86)\AutoIt3\SciTE\AutoIt3Wrapper\AutoIt3Wrapper.exe" /run /prod /ErrorStdOut /in "C:\Users\Gre90r\Desktop\AutoIt v3 Script (neu).au3" /autoit3dir "D:\Programme\AutoIt3" /UserParams
+>12:32:35 Starting AutoIt3Wrapper v.2.1.2.9    Environment(Language:0407  Keyboard:00000407  OS:WIN_7/Service Pack 1  CPU:X64 OS:X64)
-> Skipping AU3check: Current version doesn't support the AutoIt3 v 3.3.9.5+ syntax.
>Running:(3.3.10.2):D:\Program Files (x86)\AutoIt3\autoit3.exe "C:\Users\Gre90r\Desktop\AutoIt v3 Script (neu).au3"
--> Press Ctrl+Alt+F5 to Restart or Ctrl+Break to Stop
"C:\Users\Gre90r\Desktop\AutoIt v3 Script (neu).au3" (9) : ==> Array variable has incorrect number of subscripts or subscript dimension range exceeded.:
$array[5] = "-Programmabsturz"  <- Der Compiler zeigt, um welche Programmzeile es sich handelt
^ ERROR
->12:32:35 AutoIt3.exe ended.rc:1
>Exit code: 1    Time: 0.409
```

Fügen wir vor Zeile 9 eine *MsgBox()* ein, sehen wir, dass der Programmcode wirklich ausgeführt wird, bis eine Speicherverletzung erfolgt.



Drücken wir auf 'OK', läuft das Programm weiter bis in Zeile 11 der Speicher verletzt wird. Das wird uns folgendermaßen mitgeteilt.



Das Sicherheitssystem von AutoIt beendet unser Programm und lässt den Compiler eine rote Zeile auswerfen.

Sollte irgendein Programm, das wir gerade schreiben aus heiterem Himmel einfach abstürzen, ist es sehr wahrscheinlich ein Array, das eine Speicherverletzung auslöst.
Wenn man ein Array programmiert, sollte man immer zweimal hinschauen, wie damit gearbeitet wird. Arrays sind eins der wenigen Programmierelemente, die besonders fehleranfällig sind. Der Umgang mit Arrays sollte gut geübt sein.

7.1) Der Sinn hinter Arrays

Exkurs:
Das war bis jetzt ein ziemlich langes Kapitel. Wozu braucht man aber Arrays? Wir wissen ja wie man Variablen erschafft und wie man Werte abspeichert und abruft.

Arrays dienen als Struktur. Mehrere Variablen können zu einem "Thema" gehören. Zum Beispiel die Position eines Spielers in einem Spiel. Dafür könnten wir Variablen wie $playerX und $playerY erschaffen wenn es 2D ist. Wenn es aber 3D ist kommt noch eine Variable hinzu, die man einzeln kennen muss. Vielleicht hat der Spieler auch noch eine Geschwindigkeit, mit der er sich bewegt etc. Man sieht, dass zu einem "Thema", wie ein Spieler, in einem Programm mehrere Sachen/Variablen gehören.
Leichter wird die Angelegenheit, wenn man seine Variablen nicht immer zusammen suchen muss, wenn man irgendwann hundert Variablen in seinem Programm vorfindet.
Wir legen einfach ein Array mit dem Namen player an.

Folgender Programmcode soll eine Struktur mit Arrays zeigen.

```
1
2   ; set values for array indices to point to the element in the array which is needed
3   Dim $PLAYER_X = 0    ; x pos of the player will be the 1st element in the array
4   Dim $PLAYER_Y = 1    ; y pos of the player will be the 2nd element in the array
5   Dim $PLAYER_SPEED = 2    ; speed of the player will be the 3rd element in the array
6   Dim $PLAYER_HEALTH = 3   ; health will be the 4th element
7
8   ; size of array depending on the number of player variables
9   Dim $SIZE = 4
10
11  ; create player array
12  Dim $player[$SIZE]
13
14  ; set player values
15  $player[$PLAYER_X] = 20
16  $player[$PLAYER_Y] = 80
17  $player[$PLAYER_SPEED] = 50
18  $player[$PLAYER_HEALTH] = 100
19
20  ; get player values
21  MsgBox(0, "player stats", "player position X = "&$player[$PLAYER_X])
22  MsgBox(0, "player stats", "player position Y = "&$player[$PLAYER_Y])
23  MsgBox(0, "player stats", "player speed = "&$player[$PLAYER_SPEED])
24  MsgBox(0, "player stats", "player health = "&$player[$PLAYER_HEALTH])
25
26  Exit
```

Alles bezüglich des Spielers steht in dem Array $player[]. Zeile 3 bis 6 ist Teil der Struktur, damit man das Array besser benutzen kann. Soll heißen, man muss nicht die rohen Indices wählen wie $player[0]. Was steht eigentlich an Index 0. Wenn man es nicht weiß, muss man es im Programmcode nachschauen. Wir wissen durch Nachschauen, dass an Index 0, die X Position des Spielers abgelegt ist. Wieso legen wir dann nicht eine Variable an, die $PLAYER_X heißt und sich den Index 0 merkt. Genau das wurde nämlich gemacht und ist eine Art, Struktur in das Arbeiten mit Arrays zu bringen. Das Arbeiten mit Arrays wird leichter und zusätzlich wird der Programmcode für andere Programmierer besser lesbar. Mit $player[$PLAYER_X] kann man sich denken auf was man zugreifen will, mit $player[0] hat man keine Chance zu wissen, auf was gerade zugegriffen werden soll.

Abschließend zu Arrays:
Arrays hin oder her. Sie sind eine Art Hilfsmittel. Wenn man sie benutzen möchte, kann man das gerne machen, wenn man anders besser zurecht kommt, dann lässt man sie einfach weg.

Man sollte sich aber im Hinterkopf behalten, dass manche Funktionen von AutoIt Arrays zurück geben. *InputBox()* gibt uns bspw. eine Zeichenkette zurück, mit der wir weiter arbeiten können. Gibt eine Funktion aber ein Array zurück, dann muss man nur wissen wie man an die Werte des Arrays ran kommt.

8) Finale - endlich ein richtiges Programm

Endlich ein richtiges Programm nach all dem Aufwand. Du weißt vielleicht selber noch nicht, was du an dieser Stelle kannst und was du überhaupt mit dem Wissen anstellst. Wie gesagt vielleicht. Hier ist ein Programm, das du in der Lage bist selber zu schreiben.

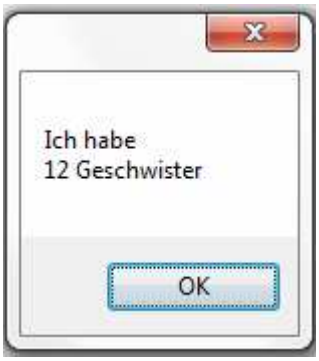
Bevor alles anfängt noch ein paar Exkurse, die wichtig sein könnten.

Exkurs: @CRLF

@CRLF, sieht kryptisch aus, ist aber leicht erklärt. Es bewirkt einen Zeilenumbruch. Es wird oft in Zeichenketten gebraucht.

Beispiel:

```
1
2   Dim $number = 12
3
4   Dim $string = "Ich habe"&@CRLF&$number&" Geschwister"
5
6   MsgBox(0, "", $string)
7
8   Exit
```



Exkurs: &

Das Kaufmanns-Und ('&') dient dazu, um Zeichenketten zu verknüpfen.

Das funktioniert so

```
Dim $string1 = "Schönes "
Dim $string2 = "Wetter "
Dim $string3 = "draußen"

Dim $totalString = $string1&$string2&$string3
```

oder so...

```
Dim $sameTotalString = $string1
$sameTotalString &= $string2
$sameTotalString &= $string3
```

'&=' kann man sich auf der gleichen Ebene wie '+=' vorstellen. 5 += 2 ist 7. '+=' addiert und weist gleich zu. '&=' verknüpft das was rechts steht, mit dem was links steht, also verknüpfen und gleich zuweisen.

$totalString und $sameTotalString sind identisch.

Der Rätselmeister ist zurück

```
1
2   ; vindov specific variables
3   Dim $TITLE = "Der Rätselmeister ist zurück"
4   Dim $WIDTH = 800
5   Dim $HEIGHT = 600
6
7   ; this variable gets the player input
8   Dim $userInput = " "
9
10  ; the menu displayed in the InputBox()
11  Dim $menu = $TITLE&@CRLF
12  $menu &= "                    "&@CRLF&@CRLF
13  $menu &= "Treff eine Weise Auswahl. Gib unten 1, 2, 3 oder 4 ein."&@CRLF&@CRLF&@CRLF
14  $menu &= "1) ein kleiner Mathe-Test"&@CRLF&@CRLF
15  $menu &= "2) Wortsuche"&@CRLF&@CRLF
16  $menu &= "3) <GEHEIM>"&@CRLF&@CRLF&@CRLF&@CRLF
17  $menu &= "4) Programm beenden"
18
19  ; this is our main loop
20  While 1
21
22      ; everything's prepared to get in contact with the user
23      $userInput = InputBox($TITLE, $menu, "", "", $WIDTH, $HEIGHT)
24
25      ; switch the user input
26      Switch $userInput
27
28          ; user selected "ein kleiner Mathe-Test"
29          Case 1
30              $userInput = InputBox("kleiner Mathe-Test", "27 + 48?")
31
32              If $userInput == (27 + 48) Then
33                  MsgBox(0, "", "richtig")
34              Else
35                  MsgBox(0, "", "falsch. Versuch es doch noch mal.")
36              EndIf
37
```

```
; user selected "Wortsuche"
Case 2
    $userInput = InputBox("Wortsuche", "Wie nennt man eine kalte Jahreszeit? (6 Buchstaben)")

    If $userInput == "Winter" Then
        MsgBox(0, "", "richtig")
    ElseIf $userInput == "Herbst" Then
        MsgBox(0, "", "Naa. Nicht ganz")
    ElseIf $userInput == "Frühling" Then
        MsgBox(0, "", "falsch. 6 Buchstaben hat das Wort")
    ElseIf $userInput == "Sommer" Then
        MsgBox(0, "", "auf keinen Fall")
    Else
        MsgBox(0, "", "falsch. Versuch es doch noch mal")
    EndIf

; user selected "<GEHEIM>"
Case 3
    $userInput = InputBox("GEHEIM", "Dieser Abschnitt des Programms verlangt ein Password."&@CRLF&"Bitte gib das Passwort ein.", "", "*")

    If $userInput == "sehrsehrgeheimespasswort" Then

        Dim $ARRAY_SIZE = 5
        Dim $userInformation[$ARRAY_SIZE]

        $userInformation[0] = "Ich kenne deinen Namen -> "&@ComputerName
        $userInformation[1] = "Ich kenne deine lokale LAN-Adresse -> "&@IPAddress1
        $userInformation[2] = "Ich kenne deine Bildschirmauflösung -> "&@DesktopWidth&" x "&@DesktopHeight&" @ "&@DesktopRefresh&" "&"Hertz"
        $userInformation[3] = "Du verwendest folgendes Betriebssystem -> "&@OSVersion
        $userInformation[4] = "Heute ist der "&@MDAY&"."&@MON&"."&@YEAR

        For $i = 0 To $ARRAY_SIZE - 1 Step 1
            MsgBox(0, "GEHEIM", $userInformation[$i])
        Next

    Else
        MsgBox(16, "GEHEIM", "Zugriff verweigert!")
    EndIf

; user selected "Programm beenden"
Case 4
    ; jumps out of the infinite loop
    ExitLoop
```

```
        ; user input did not match any of our options
        Case Else
            MsgBox(64, $TITLE, "Falsche Eingabe!")
            ContinueLoop

    EndSwitch

WEnd

MsgBox(0, $TITLE, "Programm wird beendet")

Exit
```

So. In diesem Programm kam jetzt so gut wie alles dran, wovon ich die ganze Zeit geredet hab'. Du kannst es selber ausführen, wenn du den unten stehenden Code kopierst.
Wenige Sachen kamen neu dazu, die ich noch erklären möchte. Alle Befehle, die mit '@' anfangen, sind so genannte Makros. AutoIt hat deren Funktion schon vordefiniert. Manche bewirken etwas in Zeichenketten wie @CRLF und manche tragen Informationen in sich wie @ComputerName. Manche enthalten interessante Informationen, wie der geheime Bereich des Programms zeigt.
Wenn du wissen möchtest, ob du alles verstanden hast, versuche das Programm selber zu programmieren. Bedenke aber, dass es für ein Problem viele Lösung gibt. Dein Code sieht wahrscheinlich am Ende ganz anders aus als meiner.

Hier ist der Programmcode zum Copy&Paste:

```
; window specific variables
Dim $TITLE = "Der Rätselmeister ist zurück"
Dim $WIDTH = 800
Dim $HEIGHT = 600

; this variable gets the player input
Dim $userInput = " "

; the menu displayed in the InputBox()
Dim $menu = $TITLE&@CRLF
$menu &= "______________________"&@CRLF&@CRLF
$menu &= "Treff eine Weise Auswahl. Gib unten 1, 2, 3 oder 4 ein."&@CRLF&@CRLF&@CRLF
$menu &= "1) ein kleiner Mathe-Test"&@CRLF&@CRLF
$menu &= "2) Wortsuche"&@CRLF&@CRLF
$menu &= "3) <GEHEIM>"&@CRLF&@CRLF&@CRLF&@CRLF
$menu &= "4) Programm beenden"

; this is our main loop
While 1

        ; everything's prepared to get in contact with the user
        $userInput = InputBox($TITLE, $menu, "", "", $WIDTH, $HEIGHT)

        ; switch the user input
        Switch $userInput

                ; user selected "ein kleiner Mathe-Test"
                Case 1
                        $userInput = InputBox("kleiner Mathe-Test", "27 + 48?")

                        If $userInput == (27 + 48) Then
                                MsgBox(0, "", "richtig")
                        Else
                                MsgBox(0, "", "falsch. Versuch es doch noch mal.")
                        EndIf

                ; user selected "Wortsuche"
                Case 2
                        $userInput = InputBox("Wortsuche", "Wie nennt man eine kalte Jahreszeit? (6
Buchstaben)")

                        If $userInput == "Winter" Then
                                MsgBox(0, "", "richtig")
                        ElseIf $userInput == "Herbst" Then
                                MsgBox(0, "", "Naa. Nicht ganz")
                        ElseIf $userInput == "Frühling" Then
                                MsgBox(0, "", "falsch. 6 Buchstaben hat das Wort")
                        ElseIf $userInput == "Sommer" Then
                                MsgBox(0, "", "auf keinen Fall")
                        Else
                                MsgBox(0, "", "falsch. Versuch es doch noch mal")
                        EndIf

                ; user selected "<GEHEIM>"
                Case 3
                        $userInput = InputBox("GEHEIM", "Dieser Abschnitt des Programms verlangt ein
Password."&@CRLF&"Bitte gib das Passwort ein.", "", "*")

                        If $userInput == "sehrsehrgeheimespasswort" Then

                                Dim $ARRAY_SIZE = 5
                                Dim $userInformation[$ARRAY_SIZE]

                                $userInformation[0] = "Ich kenne deinen Namen -> "&@ComputerName
                                $userInformation[1] = "Ich kenne deine lokale LAN-Adresse ->
"&@IPAddress1
                                $userInformation[2] = "Ich kenne deine Bildschirmauflösung ->
"&@DesktopWidth&" x "&@DesktopHeight&" @ "&@DesktopRefresh&" "&"Hertz"
                                $userInformation[3] = "Du verwendest folgendes Betriebssystem ->
"&@OSVersion
                                $userInformation[4] = "Heute ist der "&@MDAY&"."&@MON&"."&@YEAR

                                For $i = 0 To $ARRAY_SIZE - 1 Step 1
                                        MsgBox(0, "GEHEIM", $userInformation[$i])
                                Next

                        Else
                                MsgBox(16, "GEHEIM", "Zugriff verweigert!")
                        EndIf

                ; user selected "Programm beenden"
                Case 4
                        ; jumps out of the infinite loop
                        ExitLoop

                ; user input did not match any of our options
                Case Else
                        MsgBox(64, $TITLE, "Falsche Eingabe!")
                        ContinueLoop

        EndSwitch

WEnd

MsgBox(0, $TITLE, "Programm wird beendet")

Exit
```