Tutorial

Excel-UDF erstellen

UDF

User **D**efined **F**unction

Benutzerdefinierte Funktion

Hermann Baum

Stand: 27.03.2022

mail@hermann-baum.de

https://hermann-baum.de/excel/hbSort/de/xverweis.php

Inhaltsverzeichnis

Vorwort

In Excel gibt es die vordefinierten Funktionen wie z. B. `SUMME`, `WENN`, `SVERWEIS`, die man in den Formeln einer Zelle verwenden kann. Darüber hinaus ist es möglich, eigene Funktionen mit speziellen Aufgaben hinzuzufügen, die sog. UDFs (**U**ser **D**efined **F**unctions).

Eine der Möglichkeiten, dies zu verwirklichen, ist es, sie mit Hilfe der Programmiersprache VBA (Visual Basic for Applications) zu programmieren. Was dabei zu beachten ist und welche Schwierigkeiten man dabei zu überwinden hat, darum geht es in diesem Tutorial.

Erfahrung im Umgang mit VBA wird vorausgesetzt.

Dieses Tutorial bezieht sich auf die UDF mit dem Namen ‚**XVerweis2**'.
Eine Exceldatei mit dem vollständigen VBA-Code dieser UDF kann man auf dieser Webseite herunterladen:

https://hermann-baum.de/excel/hbSort/de/xverweis.php

Einleitung

Grundsätzlich kann jede in einem Modul programmierte Funktion, die nicht als ‚Private' deklariert ist, in einer Zelle des Arbeitsblattes innerhalb einer Formel verwendet werden – so wie die vordefinierten Funktionen auch. Damit sie jedoch auch Ergebnisse liefern und nicht nur den Fehlercode #WERT zurückgeben, müssen einige Einschränkungen beachtet werden.

Einschränkungen:

1. Die Funktionsnamen dürfen nicht mit den Namen der vordefinierten Funktionen kollidieren
2. Die Funktionen dürfen nur Werte berechnen und zurückgeben; sie dürfen keine Aktionen beinhalten wie z. B. Änderungen von Zellinhalten oder Änderungen von Zellformatierungen oder anderer Eigenschaften der Excel-Objekte
3. Arrays eines festgelegten Datentyps können weder als Parameter übergeben noch als Funktionsergebnis zurückgegeben werden; dazu muss man den Datentyp ‚Variant' verwenden

Kleines Einführungsbeispiel: Der erste Montag im Jahr

Eine UDF, die als Parameter eine Jahreszahl erwartet und das Datum des ersten Montags im Neuen Jahr zurückgibt, könnte so aussehen:

```
Function ErsterMontag(Jahr As Long) As Date
    Dim Neujahrstag As Date
    Dim tagNr As Long
    Neujahrstag = DateSerial(Jahr, 1, 1)
    tagNr = Neujahrstag Mod 7     'Samstag = 0
    If tagNr < 3 Then
        ErsterMontag = Neujahrstag + 2 - tagNr
    Else
        ErsterMontag = Neujahrstag + 9 - tagNr
    End If
End Function
```

Eine solche Funktion kann zum einen innerhalb des VBA-Codes von anderen Funktionen oder Prozeduren aufgerufen werden, zum anderen kann sie auch in einer Formel auf dem Arbeitsblatt verwendet werden, z. B. =ErsterMontag($B$4).

Mittelpunkt des Tutorials: Die Funktion XVERWEIS2

Im Mittelpunkt dieses Tutorials steht das Erstellen einer komplexeren UDF. Es handelt sich um eine Funktion mit dem Namen **XVERWEIS2**. Sie erweitert die bereits vielfältigen Möglichkeiten der Excel-Funktion `XVERWEIS`.

Sie bietet **zusätzlich** zu `XVERWEIS` die folgenden drei Fähigkeiten:

- ✓ Suche nach **mehreren** Suchkriterien
- ✓ Einsatz von **Wildcards** in den Suchkriterien
- ✓ Rückgabe **aller** Übereinstimmungen, nicht nur der ersten (optional)

Die Betrachtungen über die Funktion `XVERWEIS2` gliedern sich in vier Kapitel:

1. Syntax der Funktion
2. Auswertung der Parameterliste
3. Durchführung der Suche
4. Aufbereitung des Rückgabewertes

1. Syntax der Funktion

Die ersten sechs Parameter der Funktion `XVERWEIS2` stimmen mit denen der Funktion `XVERWEIS` überein. Nach dem sechsten Parameter können weitere Paare, bestehend aus Suchkriterium und Suchmatrix, folgen.

Syntax:

```
= XVerweis2 ( Suchkriterium1; Suchmatrix1; Rückgabematrix; [wenn_nicht_gefunden];
  [Vergleichsmodus]; [Suchmodus];
  [Suchkriterium2]; [Suchmatrix2]; [Suchkriterium3]; [Suchmatrix3]; ... )
```

Parameter	Erläuterung
Suchkriterium 1	1. Suchkriterium
Suchmatrix 1	Array oder Bereich, in dem das 1. Suchkriterium gesucht wird (muss eindimensional, d.h. eine einzelne Zeile oder einzelne Spalte, sein)
Rückgabematrix	Array oder Bereich, aus dem gefundene Werte zurückgegeben werden
wenn_nicht_gefunden (optional)	Text, der an Stelle des Errorcodes #NV zurückgegeben wird, wenn nichts gefunden wurde
Vergleichsmodus (optional)	Typ der Übereinstimmung beim Suchen: 0: Suche nach genauer Übereinstimmung (Standard) **1**: Ungültig (erzeugt den Error #WERT) **-1**: Ungültig (erzeugt den Error #WERT) 2: Suche mit Wildcard-Symbolen, die Wildcards *, ?, ~ können für die Suche eingesetzt werden
Suchmodus (optional)	Suchmodus: 1: Normale Suchreihenfolge beginnend mit dem ersten Element; die erste gefundene Übereinstimmung wird zurückgegeben (Standard) -1: Umgekehrte Suchreihenfolge beginnend mit dem letzten Element; die erste gefundene Übereinstimmung wird zurückgegeben 2: Normale Suchreihenfolge beginnend mit dem ersten Element; alle Übereinstimmungen werden zurückgegeben -2: Umgekehrte Suchreihenfolge beginnend mit dem letzten Element; alle Übereinstimmungen werden zurückgegeben
Suchkriterium 2 (optional)	2. Suchkriterium
Suchmatrix 2 (optional)	Array oder Bereich, in dem das 2. Suchkriterium gesucht wird (eindimensional - einzelne Spalte oder einzelne Zeile)
Suchkriterium 3 (optional)	3. Suchkriterium
Suchmatrix 3 (optional)	Array oder Bereich, in dem das 3. Suchkriterium gesucht wird (eindimensional - einzelne Spalte oder einzelne Zeile)
usw.	. . .

Suchkriterien, die nicht spezifiziert sind (nur Semikolon) oder den leeren String enthalten, haben keine Auswirkung auf die Suche.

2. Auswertung der Parameterliste

Als ersten Schritt muss man die Parameterliste der Zellfunktion `XVERWEIS2` in eine Parameterliste der VBA-Funktion `XVerweis2` umsetzen. Der zweite Schritt besteht dann aus dem Auslesen der Parameterwerte in entsprechende VBA-Variablen.

2.0 Parameterliste in VBA

In VBA gibt es mehrere mögliche Strukturen für Parameterlisten.

1. Nur Pflichtparameter
   z. B. `Function Name (p1 As long, p2 As String) As Double`
2. Pflichtparameter und eine feste Anzahl von optionalen Parametern
   z. B.
   ```
   Function Name (p1 As long, p2 As String, Optional p3 As long = 1, _
                  Optional p4 As long = 0)
   ```
3. Pflichtparameter und eine unbestimmte Anzahl von optionalen Parametern
   z. B. `Function Name (p1 As long, p2 As String, ParamArray arg() As Variant) As String`

In allen drei Fällen kann die Anzahl der Pflichtparameter auch null sein, so dass man auch von sechs verschiedenen Fällen reden kann.

Eine Mischung aus einzelnen optionalen Parametern und dem ParamArray ist nicht möglich. Das ParamArray muss vom Typ Variant sein.

Da bei der Funktion `XVERWEIS2` analog zur Funktion `XVERWEIS` der erste Parameter bereits ausgelassen werden kann, bleibt uns nur der Fall 6: die Parameterliste besteht nur aus dem ParamArray.

Die Funktion hat also in VBA folgende Kopfzeile:
```
Function XVerweis2(ParamArray arg() As Variant)
```
Der Rückgabetyp ist nicht angegeben, also Variant. Dies muss so sein, da diese Funktion verschiedene Datentypen zurückgibt: Bereiche, Arrays oder einzelne Werte – je nach Situation.

Die erste Anweisung lautet:
```
argCnt = UBound(arg)
```

Die Anzahl der Parameter wird in der Variablen `argCnt` gespeichert. Da das ParamArray nullbasiert ist, wird hier die Zahl 4 gespeichert, wenn 5 Parameter angegeben wurden.

Mindestens 3 Parameter müssen angegeben sein. Deshalb sorgt die nächste Code-Zeile dafür, dass bei weniger als 3 Parametern ein Error-Code zurückgegeben wird:
```
If argCnt < 2 Then EXIT_BY_ERROR
```

Die kleine Hilfsprozedur `EXIT_BY_ERROR` hat den Vorteil, dass sie in verschiedenen UDFs verwendet werden kann. Sie erzeugt absichtlich einen Fehler, so dass die weitere Ausführung der Funktion abgebrochen und der Fehlercode #WERT zurückgegeben wird.

```
Private Sub EXIT_BY_ERROR()
    Dim errorArr() As Long
    errorArr(0) = 1   'exit by #VALUE error
End Sub
```

Da hier die Redim-Anweisung fehlt, wird der Error ‘Index außerhalb des gültigen Bereichs‘ erzeugt. Dadurch dass die Funktion `XVerweis2` als Zellfunktion aufgerufen wurde, erscheint keine Fehlermeldung, sondern die Zellfunktion antwortet mit dem Fehlercode #WERT bzw. #VALUE – je nach Landessprache.

Die Anzahl der Suchkriterien ergibt sich aus der Parameteranzahl. Sie wird mit dem folgenden Code ermittelt:

```
If argCnt < 6 Then
    critCnt = 1
Else
    critCnt = (argCnt - 4) \ 2 + 1   'Erläuterung des Rechenterms siehe Kap. 2.7
End If
```

Die verschiedenen Paare bestehend aus Suchkriterium und Suchmatrix sollen in den beiden Arrays `critArr` und `lookArr` gespeichert werden. Sie werden deshalb an dieser Stelle entsprechend dimensioniert:

```
ReDim critArr(1 To critCnt)
ReDim lookArr(1 To critCnt)
```

2.1 Auswertung des 1. Parameters

Das Auslesen des ersten Parameters ist einfach:

```
If IsMissing(arg(0)) Then
    critArr(1) = ""
Else
    critArr(1) = arg(0)
End If
```

Mit der Funktion `IsMissing` kann nachgeprüft werden, ob ein Parameter angegeben oder ob nur ein Semikolon gesetzt wurde. Im zweiten Fall wird als 1. Kriterium der leere String eingetragen. Kriterien, die aus dem leeren String bestehen, haben bei der Funktion `SVERWEIS2` keine Auswirkung auf die Suche.

Die verschiedenen Suchkriterien sollen in dem Array `critArr` gespeichert werden. Deshalb wird dieser erste Parameter in der Variablen `critArr(1)` gespeichert.

2.2 Auswertung des 2. Parameters

Der zweite Parameter muss ein Bereich oder ein Array sein. Diese Prüfung erledigt die Anweisung
`If Not IsArray(arg(1)) Then EXIT_BY_ERROR`

Die Funktion `IsArray` gibt auch dann den Wert FALSE zurück, wenn die Adresse einer einzelnen Zelle oder ein Array, das nur aus einem Element besteht, als Parameter angegeben wird.

Mit der nächsten Anweisung
`lkArr = arg(1)`
werden die Werte der Variablen `arg(1)` in die Variable `lkArr` kopiert.

Dies ist insofern ein kritischer Punkt, als wir beachten müssen, dass `arg(1)` entweder ein Bereich oder ein Array sein kann. Sicherlich wird ein Anwender in der Regel einen Bereich als Parameter angeben und nur in Spezialfällen ein Array. Jedoch gibt es ja noch die Fälle, bei denen für diesen Parameter Formeln eingesetzt werden, die entweder Bereiche zurückgeben – wie z. B. `BEREICH.VERSCHIEBEN` – oder vielfach auch Arrays von Werten.

Die Variable `lkArr` ist zunächst eine nicht initialisierte Variable vom Typ Variant. Bei dem oben genannten Kopiervorgang sind nun drei Fälle zu betrachten:

1. `arg(1)` ist ein Bereich.
   In diesem Fall wird `lkArr` zu einem Array mit den gleichen Dimensionen wie `arg(1)`, also ein zweidimensionales Array mit 1 Zeile und n Spalten oder mit n Zeilen und 1 Spalte.
2. `arg(1)` ist ein Array bestehend aus n Zeilen und einer Spalte.
   Auch hier hat `lkArr` die gleichen Dimensionen wie `arg(1)`.

3. `arg(1)` ist ein Array bestehend aus 1 Zeile und n Spalten.
   Dies ist der kritische Fall, weil nun die Variable `lkArr` nur noch eine Dimension besitzt und die Abfrage der 2. Dimension mit `UBound(lkArr, 2)` zu einem Runtime-Error führt.

Warum ein einspaltiges Array zwei Dimensionen hat, ein einzeiliges Array dagegen nur eine Dimension, weiß ich nicht. Mir kommt es vor wie ein Microsoft-Bug.

Da dieser Umstand an mehreren Stellen berücksichtigt werden muss, kommt als Workaround die kleine Prozedur `repairArray` zum Einsatz.

```
Private Sub repairArray(ByRef arr As Variant)
    Dim res As Variant
    Dim uBnd As Long
    Dim maxCol As Long
    Dim col As Long

    If Not IsObject(arr) Then      'arr is an array, not a range
On Error GoTo UBoundError          'Workaround for a single row array
        uBnd = UBound(arr, 2)
        GoTo GoOn                  'no error
UBoundError:
        maxCol = UBound(arr, 1)
        ReDim res(1 To 1, 1 To maxCol)
        For col = 1 To maxCol
            res(1, col) = arr(col)
        Next
        arr = res
        Resume GoOn
GoOn:
On Error GoTo 0
    End If
End Sub
```

Sie prüft, ob es sich beim betreffenden Parameter um den kritischen 3. Fall handelt. Wenn bei der Anweisung
`uBnd = UBound(arr, 2)`
ein Fehler ausgelöst wird, wird der Fehler abgefangen und das Array in ein zweidimensionales Array (1 Zeile und n Spalten) umgewandelt.

Nun können die Dimensionen ermittelt werden:

```
maxrowLook = UBound(lkArr, 1)
maxcolLook = UBound(lkArr, 2)
```

Wenn es sich nicht um ein eindimensionales Array handelt, gibt die Funktion `XVERWEIS2` den Fehlercode zurück:

```
If maxrowLook > 1 And maxcolLook > 1 Then EXIT_BY_ERROR
```

In der Variablen `byRow` wird gespeichert, ob es sich um einen Suchvorgang waagerecht in der Zeile (`byRow = TRUE`) oder senkrecht in der Spalte (`byRow = FALSE`) handelt:

```
byRow = (maxcolLook > 1)
```

Die Variable `maxInd` enthält die Anzahl der Zeilen bzw. die Anzahl der Spalten – je nachdem, ob waagerecht oder senkrecht gesucht wird:

```
If byRow Then
    maxInd = maxcolLook
Else
    maxInd = maxrowLook
End If
```

Für jedes Suchkriterium gibt es in der Parameterliste eine zugehörige Suchmatrix. So wie die Suchkriterien in einem Array namens `critArr` gespeichert werden, sollen die Suchmatrices analog in einem Array namens `lookArr` gespeichert werden. Die Variable `lookArr` ist somit ein Array von Arrays.

In dem (selteneren) Fall einer horizontalen Suche wird die Suchmatrix von einem einzeiligen Array in ein einspaltiges Array umgewandelt (Konvertierung der $1 \times n$-Matrix in eine $n \times 1$-Matrix). Dies geschieht durch den folgenden Code-Abschnitt:

```
If byRow Then
    ReDim arr(1 To maxInd, 1 To 1)
    For ind = 1 To maxInd
        arr(ind, 1) = lkArr(1, ind)
    Next
    lookArr(1) = arr
Else
    lookArr(1) = lkArr
End If
```

Auf diese Weise werden die Suchkriterien und Suchmatrices in einem einheitlichen Format an die eigentlichen Suchroutinen übergeben.

2.3 Auswertung des 3. Parameters

Zunächst wird mit Hilfe der Funktion `isArray()` wieder ausgeschlossen, dass hier ein Einzelwert oder die Adresse einer einzelnen Zelle eingetragen wurde:

```
If Not IsArray(arg(2)) Then EXIT_BY_ERROR
```

Danach erfordert dieser Parameter aus ganz anderem Grund ebenfalls eine besondere Behandlung. Microsoft hat für seine Funktion `XLOOKUP` spezifiziert, dass in den Fällen, wo es möglich ist, nicht ein Array von Werten zurückgegeben wird, sondern der Bereich, der diese Werte enthält. Aus VBA-Sicht wird also in diesen Fällen kein Array zurückgegeben, sondern ein Range-Objekt.

Dies hat folgenden Zusatzeffekt: Ein Funktionsaufruf mit der Funktion `XLOOKUP` kann als Parameter in eine Funktion eingesetzt werden, die einen Bereich als Parameter erwartet (siehe Beispiel am Ende des Tutorials).

In dem folgenden Code-Abschnitt wird deshalb mit Hilfe der Funktion `isObject()` geprüft, ob es sich beim dritten Parameter um ein Range-Objekt oder um ein Array von Werten handelt:

```
If IsObject(arg(2)) Then
    Set retArr = arg(2)
    maxrowRet = retArr.Rows.Count
    maxcolRet = retArr.Columns.Count
Else
    retArr = arg(2)
    Call repairArray(retArr)
    maxrowRet = UBound(retArr, 1)
    maxcolRet = UBound(retArr, 2)
End If
```

Im ersten Fall werden die Werte nicht einfach in die Variable `retArr` kopiert, sondern mit der Anweisung
`Set retArr = arg(2)`
zeigt die Variable `retArr` auf dasselbe Objekt wie `arg(2)`. Sie besitzt damit den Typ ‚Range'.

In den Variablen `maxrowRet` und `maxcolRet` werden die Dimensionen des dritten Parameters gespeichert. Die nächsten beiden Anweisungen erzeugen einen Error, wenn die Länge der ersten Suchmatrix nicht mit der entsprechenden Dimension der Rückgabematrix übereinstimmt:

```
If byRow And (maxcolLook <> maxcolRet) Then EXIT_BY_ERROR
If Not byRow And (maxrowLook <> maxrowRet) Then EXIT_BY_ERROR
```

2.4 Auswertung des 4. Parameters

Es kann sein, dass nach dem 3. Parameter nichts mehr folgt in der Parameterliste. In diesem Fall hat die Variable `argCnt` den Wert 2. Deshalb beginnt die Auswertung mit der Abfrage `If argCnt >= 3`.

Der VBA-Code für die Auswertung des 4. Parameters lautet:

```
If argCnt >= 3 Then
    If IsMissing(arg(3)) Then
        notFnd = CVErr(xlErrNA)
    Else
        notFnd = arg(3)
    End If
Else
    notFnd = CVErr(xlErrNA)
End If
```

Da der Variablen `notFnd` zum einen ein Error-Typ zugewiesen werden kann, zum anderen aber auch ein String, muss sie unbedingt vom Typ Variant sein, sonst tritt u. U. ein Laufzeitfehler auf.

Der Ausdruck `CVErr(xlErrNA)` gibt den Error #NV zurück. Wenn der 4. Parameter nicht angegeben wird, erscheint im Fall von null Treffern der Errorcode #NV in den Zellen. Ansonsten erscheint der Wert, der für diesen Parameter eingesetzt wurde. An Stelle des Ausdrucks `CVErr(xlErrNA)` einen festen String (z. B. „#NV“ oder „#N/A“ im Englischen) auszugeben wäre die schlechtere Lösung, da die Funktion CVErr den Error #NV in der jeweiligen Landessprache ausgibt.

2.5 Auswertung des 5. Parameters

Hier wird dafür gesorgt, dass der Standardwert 0 ist und andere Werte als 0 oder 2 zu einem Error führen:

```
If argCnt >= 4 Then
    If IsMissing(arg(4)) Then
        mMode = 0
    Else
        mMode = arg(4)
    End If
Else
    mMode = 0
End If
If mMode <> 0 And mMode <> 2 Then EXIT_BY_ERROR
```

2.6 Auswertung des 6. Parameters

Der Standardwert ist 1 und andere Werte als 1, -1, 2 oder -2 führen zu einem Error.

```
If argCnt >= 5 Then
    If IsMissing(arg(5)) Then
        sMode = 1
    Else
        sMode = arg(5)
    End If
Else
    sMode = 1
End If
If sMode <> 1 And sMode <> 2 And sMode <> -1 And sMode <> -2 Then EXIT_BY_ERROR
```

2.7 Auswertung der restlichen Parameter

Nach dem sechsten Parameter können noch eine unbestimmte Anzahl an Paaren, bestehend aus Suchkriterium und Suchmatrix, folgen. Die Auswertung, d. h. Überprüfung und Speicherung in den Variablen `critArr` bzw. `lookArr` erfolgt daher in einer Schleife.

```
    If argCnt >= 6 Then
        For argNr = 6 To argCnt
            critNr = (argNr - 4) \ 2 + 1
            If argNr Mod 2 = 0 Then   'lookup criterion
                If IsMissing(arg(argNr)) Then
                    critArr(critNr) = ""
                Else
                    critArr(critNr) = arg(argNr)
                End If
            Else         'lookup array
                If Not IsArray(arg(argNr)) Then EXIT_BY_ERROR
                lkArr = arg(argNr)
                Call repairArray(lkArr)
                maxrowLook = UBound(lkArr, 1)
                maxcolLook = UBound(lkArr, 2)
                If maxrowLook > 1 And maxcolLook > 1 Then EXIT_BY_ERROR

                If byRow Then
                    ReDim arr(1 To maxInd, 1 To 1)
                    For ind = 1 To maxInd
                        arr(ind, 1) = lkArr(1, ind)
                    Next
                    lookArr(critNr) = arr
                Else
                    lookArr(critNr) = lkArr
                End If
            End If
        Next
    End If
```

Die Anzahl der Argumente wurde ja bereits ermittelt und in der Variablen `argCnt` gespeichert (siehe Kap. 2.0). Da das ParamArray nullbasiert ist, beginnt die FOR-Schleife mit 6, also dem 7. Parameter.

Mit der Anweisung
`critNr = (argNr - 4) \ 2 + 1`
wird die laufende Nummer der Kriterien ermittelt. Die ersten beiden Parameter haben im Array `arg()` die Indices 0 und 1. Sie bekommen in den Arrays `critArr` und `lookArr` den Index 1 (1. Kriterienpaar). Das zweite Kriterienpaar hat, falls vorhanden, in der Parameterliste `arg()` die Indizes 6 und 7, das dritte Kriterienpaar die Indizes 8 und 9 usw.

Das heißt, die Parameter-Indizes 6 und 7 müssen zum Kriterium-Index 2 führen, die Parameter-Indizes 8 und 9 zum Kriterien-Index 3 usw. Dies erreicht man, wenn man z. B. vom Parameter-Index 7 die Zahl 4 subtrahiert, das Ergebnis, die Zahl 3, mit der Ganzzahldivision durch 2 dividiert (ergibt 1) und die Zahl 1 addiert. Das Resultat der Rechnung ist 2.
Auf dieselbe Weise führen die Parameter-Indizes 8 und 9 zu dem Kriterien-Index 3.

Die Auswertung der Suchkriterien erfolgt analog zur Auswertung des ersten Suchkriteriums (siehe Kap. 2.1) und die Auswertung der Suchmatrices analog zur Auswertung der ersten Suchmatrix (siehe Kap. 2.2).

3. Durchführung der Suche

Die beiden Parameter Vergleichsmodus (0 oder 2) und Suchmodus (1, -1, 2, oder -2) ermöglichen acht verschiedene Fälle.

Vergleichsmodus (optional)	Typ der Übereinstimmung beim Suchen: 0: Suche nach genauer Übereinstimmung (Standard) 2: Suche mit Wildcard-Symbolen, die Wildcards *, ?, ~ können für die Suche eingesetzt werden
Suchmodus (optional)	Suchmodus: 1: Normale Suchreihenfolge beginnend mit dem ersten Element; die erste gefundene Übereinstimmung wird zurückgegeben (Standard) -1: Umgekehrte Suchreihenfolge beginnend mit dem letzten Element; die erste gefundene Übereinstimmung wird zurückgegeben 2: Normale Suchreihenfolge beginnend mit dem ersten Element; alle Übereinstimmungen werden zurückgegeben -2: Umgekehrte Suchreihenfolge beginnend mit dem letzten Element; alle Übereinstimmungen werden zurückgegeben

Die Suche nach Übereinstimmungen läuft nach folgendem Plan ab:

Wenn es sich beispielsweise um eine senkrechte Suche handelt (Suche in der Spalte), wird zunächst geprüft, ob alle Suchkriterien mit dem jeweils ersten Element der entsprechenden Suchmatrix übereinstimmen (Übereinstimmung aller Suchkriterien in der ersten Zeile). Nur bei Übereinstimmung aller Suchkriterien wird diese Zeile als Treffer gewertet und die Zeilennummer in einem „Treffer-Array“ (Variable `indArr`) gespeichert.

In den Suchmodi 1 und -1 wird die Suche nach dem ersten gefundenen Treffer abgebrochen. In den Suchmodi 2 und -2 wird bis zum Ende der Liste weitergesucht und die Zeilennummern aller Treffer werden im Array `indArr` gespeichert.

Aus Performance-Gründen werden an Stelle einer einzigen Suchroutine vier einzelne Suchroutinen mit ähnlichem VBA-Code verwendet, die auf die vier Suchmodi zugeschnitten sind.

```
If sMode = 1 Then
    indArr = SeqLookupAsc2Break(lookArr, critArr, mMode)
ElseIf sMode = -1 Then
    indArr = SeqLookupDesc2Break(lookArr, critArr, mMode)
ElseIf sMode = 2 Then
    indArr = SeqLookupAsc2All(lookArr, critArr, mMode)
Else    'sMode = -2
    indArr = SeqLookupDesc2All(lookArr, critArr, mMode)
End If
```

Innerhalb jeder dieser vier Suchroutinen wird zwischen `mMode = 0` (exakte Suche) und `mMode = 2` (Suche mit Wildcards) unterschieden:

```
Function SeqLookupAsc2Break(lookArr As Variant, critArr As Variant, mMode As Long)
    ...
    If mMode = 0 Then
    ...
    Else  'mMode = 2
    ...
    End If
    SeqLookupAsc2Break = indArr
End Function
```

Der folgende Code-Ausschnitt zeigt den Fall der exakten Suche (Vergleichsmodus = 0) mit Rückgabe aller gefundenen Treffer (Suchmodus = 2), also Aufruf der Suchroutine `SeqLookupAsc2All`.

```
If mMode = 0 Then
    ind = 1
    k = 0
    While ind <= maxInd
        gefunden = True
        For critNr = 1 To critCnt
            If (lookArr(critNr)(ind, 1) <> critArr(critNr)) And (critArr(critNr) <> "")
            Then
                gefunden = False
                Exit For
            End If
        Next
        If gefunden Then
            k = k + 1
            indArr(k) = ind
        End If
        ind = ind + 1
    Wend
Else  'mMode = 2
```

In der While-Schleife durchläuft die Variable `ind` alle Werte von 1 bis `maxInd`. In der Variablen `maxInd` ist die Anzahl der Elemente einer Suchmatrix gespeichert. Bei einer senkrechten Suche (Suche in der Spalte) wäre das also die Anzahl der Zeilen.

Wir bleiben beim Beispiel einer senkrechten Suche:
In jeder Zeile müssen alle Suchkriterien geprüft werden. Nur wenn in einer Zeile alle Suchkriterien erfüllt sind, ist ein Treffer gefunden worden. Diese Prüfung aller Suchkriterien innerhalb einer Zeile erfolgt in der inneren FOR-Schleife. Die mit `TRUE` initialisierte Variable `gefunden` wird bei der ersten Nicht-Übereinstimmung auf `FALSE` gesetzt. Wenn sie am Ende der FOR-Schleife immer noch `TRUE` ist, liegt ein Treffer vor.

Die Zusatzbedingung
`... and (critArr(critNr) <> "")`
sorgt dafür, dass Suchkriterien, die aus dem leeren String bestehen, keine Wirkung haben.

Der ELSE-Teil (`mMode = 2`) unterscheidet sich nur in einer einzigen Zeile:
Die Prüfbedingung heißt jetzt nicht mehr
`If (lookArr(critNr)(ind, 1) <> critArr(critNr)) And (critArr(critNr) <> ""),`
sondern
`If (Not (lookArr(critNr)(ind, 1) Like critArr(critNr))) And (critArr(critNr) <> "").`

Der Like-Operator vergleicht einen String mit einem Muster (Pattern). Die Syntax lautet:
`<result> = <string> Like <pattern>.`
Hier ist darauf zu achten, dass der String links vom Like-Operator steht und das Pattern mit den Wildcards rechts davon.

Jede der vier Suchroutinen gibt ein Array mit dem Variablennamen `indArr` zurück, welches die Indizes der gefundenen Treffer enthält. Wenn nur nach dem ersten Treffer gesucht wird (`sMode = 1` oder `sMode = -1`), enthält das Array maximal einen Treffer-Index, der in der Variablen `indArr(1)` gespeichert ist.

Wurde kein Treffer gefunden, ist `indArr(1) = 0` .

Das Array `indArr` bildet die Basis für die nun folgende Aufbereitung des Rückgabewertes der Funktion XVERWEIS2.

4. Aufbereitung des Rückgabewertes

Der einfachste Fall liegt vor, wenn keine Übereinstimmung gefunden wurde. Dann wird der im 4. Parameter angegebene String bzw. der Standardwert „#NV“ zurückgegeben. Deshalb beginnt die Aufbereitung mit der folgenden IF-Anweisung:

```
If indArr(1) < 1 Then
    ret = notFnd
Else
...
End If
```

Die Variable `ret` soll auf jeden Fall den Rückgabewert enthalten. Da sie vom Typ Variant ist, kann sie einen String, einen numerischen Wert, ein Array von Werten oder auch einen gefundenen Bereich enthalten.

Ein Bereich kann dann zurückgegeben werden, wenn nach einer einzelnen Übereinstimmung gesucht wurde (Suchmodus gleich 1 oder -1) und als Rückgabematrix (3. Parameter) ein Bereich übergeben wurde (also kein Array).

Der folgende Code-Ausschnitt zeigt für den Fall, dass nach allen Übereinstimmungen gesucht wurde, wie die einzelnen gefundenen Zeilen zu einem Rückgabe-Array zusammengesetzt werden:

```
If sMode = 2 Or sMode = -2 Then
    ReDim ret(1 To maxrowRet, 1 To maxcolRet)
    If byRow Then
    ...
    Else
        For row = 1 To maxrowRet
            For col = 1 To maxcolRet
                If indArr(row) >= 1 Then
                    ret(row, col) = retArr(indArr(row), col)
                Else
                    ret(row, col) = ""
                End If
            Next
        Next
    End If
Else      'sMode = 1 or -1
```

Die Variable `ret` wird so redimensioniert, dass sie dieselben Dimensionen hat wie die Rückgabematrix (3. Parameter). Alle gefundenen Zeilen werden in das Array `ret` in der gefundenen Reihenfolge übertragen. In die restlichen Zeilen werden leere Strings geschrieben.

Wurde nur nach der ersten Übereinstimmung gesucht (Suchmodus gleich 1 oder -1), sieht der Code ein wenig anders aus:

```
Else     'sMode = 1 or -1
   If byRow Then
   ...
   Else
      If IsObject(retArr) Then
         Set ret = Range(retArr.Cells(indArr(1), 1), retArr.Cells(indArr(1), maxcolRet))
      Else
         ReDim ret(1 To 1, 1 To maxcolRet)
         For col = 1 To maxcolRet
            ret(1, col) = retArr(indArr(1), col)
         Next
      End If
   End If
End If
```

Hier ist folgendes zu beachten:
Ist die Rückgabematrix (3. Parameter) ein Bereich und kein Array, dann soll die Funktion `XVERWEIS2` den **gefundenen Bereich** zurückgeben und kein Array.

Deshalb wird geprüft, ob die Variable `retArr` ein Objekt – und somit ein Bereich (Range) – ist. Wenn ja, wird der Variablen `ret` ein Verweis auf das gefundene Range-Objekt zugewiesen:

```
Set ret = Range(retArr.Cells(indArr(1), 1), retArr.Cells(indArr(1), maxcolRet))
```

Zur Erinnerung: In der Variablen `indArr(1)` steht die Zeilennummer der gefundenen Zeile. Mit ihrer Hilfe wird ein Range-Objekt gebildet, das die gefundene Zeile enthält.

Wenn die Rückgabematrix (der 3. Parameter) kein Range-Objekt ist, werden die Werte der Trefferzeile in die Variable `ret` kopiert. Die Variable `ret` ist in diesem Fall ein zweidimensionales Array, das nur aus einer Zeile besteht – oder anders ausgedrückt: eine $1 \times n$-Matrix.

Nun könnte man meinen, dass man nur noch die abschließende Anweisung `XVerweis2 = ret` hinzuschreiben braucht. Doch dann würde die Funktion in jedem Fall ein Array von Werten zurückgeben. Wenn man ein Range-Objekt zurückgeben will, lautet die Anweisung: `Set XVerweis2 = ret` . Da dies im Fall eines Arrays wiederum zu einem Runtime-Error führen würde, benötigt man eine Fallunterscheidung:

```
If IsObject(ret) Then
    Set XVerweis2 = ret
Else
    XVerweis2 = ret
End If
```

Auf diese Weise gibt die Funktion `XVERWEIS2` analog zur Funktion `XVERWEIS` im Falle der erfolgreichen Suche nach einem einzelnen Treffer einen Bereich zurück. Diesen Bereich könnte man dann als Parameter in eine Funktion einsetzen, die zwingend einen Bereich als Parameter fordert, wie zum Beispiel die Funktion `BEREICH.VERSCHIEBEN`.

Dazu ein **Beispiel**:
Die folgende Formel ist ein gültiger Funktionsaufruf:

```
=BEREICH.VERSCHIEBEN(XVerweis2($I$25;$C$4:$C$23;$B$4:$G$23;;0;1);0;2;1;3)
```

Die Funktion `XVERWEIS2` gibt aus dem Bereich `$B$4:$G$23` eine Trefferzeile zurück, welche der Funktion `BEREICH.VERSCHIEBEN` als Basis dient.