Introduction

Excel has some manual methods, such as Advanced Filter, for getting a list of distinct items from an input range. The drawback of using such methods is that you must manually refresh the results when the input data changes. Moreover, these methods work only with ranges, not arrays of values, and, not being functions, cannot be called from worksheet cells or incorporated into array formulas. This page describes a VBA function called DistinctValues that accepts as input either a range or an array of data and returns as its result an array containing the distinct items from the input list. That is, the elements with all duplicates removed. The order of the input elements is preserved. The order of the elements in the output array is the same as the order in the input values. The function can be called from an array entered range on a worksheet (see this page for information about array formulas), or from in an array formula in a single worksheet cell, or from another VB function.

The function declaration is shown below:

Function DistinctValues(InputValues As Variant, \_

Optional IgnoreCase As Boolean = False) As Variant

You can download an example workbook or just the bas module file with the complete code.

The parameter InputValues is either a range on a worksheet or an array of values. If it is a worksheet range, the range must have exactly one column or one row. Two-dimensional ranges are not supported. If InputValues is an array, it must be a single dimensional array. Two-dimensional arrays are not supported. The parameter IgnoreCase indicates whether the comparisons should be case-sensitive or case-insensitive. If this value is True, case is ignored and abc is considered equal to ABC. If this value is False, case is taken into account and abc is consider different from ABC.

If the function is array entered into a range on a worksheet, the size of the returned array is equal to the size of the range into which the function was entered, regardless of the number of distinct elements found, and unused entries at the end of the resulting array are set to vbNullStrings. This prevents #N/A errors from appearing. Note that this differs from the default behavior of Excel's own array formulas. If the function is entered in a single cell array formula, the size of the result array is equal to the number of distinct elements from the input list. Similarly, if the function is called from another VB function, not from a worksheet cell, the result array contains only the distinct elements.

Empty elements, those with a value of vbNullString or Empty are not counted as distinct elements -- they are ignored. Thus, the array {"a","b","","","c"} has three distinct elements, a, b, and c. The empty string is ignored by the function. Spaces and zero values, however, are considered when creating the list of distinct elements.

If an array, not a range, is passed into DistinctValues, that array must not contain any Object type variables (other than Excel.Range objects) and must not contain any Null values.

SectionBreak

Examples Of Calling DistinctValues

The most common usage is to array enter the DistinctValues function into a range of cells and pass it another range of cells as the input list. For example, select cells B1:B10 type

=DistinctValues(A1:A10,FALSE)

and press CTRL SHIFT ENTER. This list of distinct values from cells A1:A10 will be returned to cells B1:B10. Unpopulated cells in B1:B10 will be filled with empty strings.

You can also use DistinctValues in an array formula. For example,

=MATCH("chip",DistinctValues(A1:A10,TRUE),0)

will return the position of the string chip in the list of distinct values from cells A1:A10.

To count the number of distinct values in a range, just pass the results of DistinctValue to the COUNT or COUNTA function:

=COUNTA(DistinctValues(A1:A10,TRUE))

In addition, the DistinctValues function may be called from other VB code, passing either a Range or an Array as the input parameter. For example,

Sub Test()

Dim InputRange As Range

Dim ResultArray As Variant

Dim Ndx As Long

Set InputRange = Range("InputValues")

ResultArray = DistinctValues(InputValues:=InputRange, IgnoreCase:=True)

If IsArray(ResultArray) = True Then

For Ndx = LBound(ResultArray) To UBound(ResultArray)

Debug.Print ResultArray(Ndx)

Next Ndx

Else

If IsError(ResultArray) = True Then

Debug.Print "ERROR: " & CStr(ResultArray)

Else

Debug.Print "UNEXPECTED RESULT: " & CStr(ResultArray)

End If

End If

End Sub

In addition to a range, the InputValues can be an array literal. For example,

=DistinctValues({"a","b","a","b","c"},TRUE)

SectionBreak

The Code For Distinct Values

The code for the DistinctValues function is shown below. It requires the NumberOfArrayDimensions, TransposeArray, and Transpose1DArray functions, all of which are listed below following the listing for DisinctValues.

You can download an example workbook or just the bas module file with the complete code.

Option Explicit

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' modDistinctValues

' By Chip Pearson, 5-November-2007, chip@cpearson.com, www.cpearson.com

' This page: www.cpearson.com/Excel/DistinctValues.apsx

'

' This module contains the DistinctValues function and supporting procedures. You

' should import the entire module into your project. The DistinctValues function

' takes in a Range or an Array as input and returns an Array containing the disinct

' values from that array of inputs.

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function DistinctValues(InputValues As Variant, \_

Optional IgnoreCase As Boolean = False) As Variant

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' DistinctValues

' This function accepts a set of values in InputValues and returns an Array

' containing the distinct items in that input set. The order of elements in the result

' array is the same as in the InputValues. InputValues may be either a Range object

' or an Array. In either case, it must be one-dimensional (in the case of a Range,

' it may be either a row or column range). If InputValues has more than one dimension,

' the function returns a #REF error. The IgnoreCase parameter indicates whether to do

' a case-sensitive or case-insensitive comparison when comparing text values. If TRUE,

' case is ignored and 'abc' is treated the same as 'ABC'. If FALSE, case is taken into

' account and 'abc' is treated differently than 'ABC'.

'

' If the function is called from a worksheet, it must be array entered (CTRL SHIFT ENTER)

' into the array of cells that will receive the resutling Distinct values. The size of

' the returned array will be the same size as the array into which the function was

' entered. The Distinct values will fill the first N cells and the remaining array entries

' will be vbNullStrings. The result is properly transposed (or not) depending on whether

' it was called from a row-range or a column-range of cells on the worksheet.

' The result array is always sized to match the size of the range into which it was

' entered, even if that array contains more entries than the InputValues range. This behavior

' differs from the standard behavior of Excel's own array functions.

'

' If the function is called by another VBA procedure, not from worksheet cells, the

' array is a single dimensional array with only enough elements to contain the Distinct

' elements. The LBound of the array is 1. The variable that receives the array of distinct

' values should be declared as a Variant:

' Dim Res As Variant

' Res = DistinctElements(MyArray,True)

'

' Empty elements, those with a value of vbNullString or Empty, are not compared. Thus,

' vbNullString and Empty are not considered values in the own right and are not counted

' amongst the Distinct Values. NULL values are not allowed in the InputValues and the

' presence of a NULL value will cause an #NULL error, If there is an Object type variable

' in the InputValues other than a Range object, a #VALUE error will be returned.

'

' String representations of numbers are considered the same as numbers, so 2 and "2"

' are not distict values.

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim ResultArray() As Variant

Dim UB As Long

Dim TransposeAtEnd As Boolean

Dim N As Long

Dim ResultIndex As Long

Dim M As Long

Dim ElementFoundInResults As Boolean

Dim NumCells As Long

Dim ReturnSize As Long

Dim Comp As VbCompareMethod

Dim V As Variant

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' Set the text comparison value to be used by StrComp based on

' the setting of IgnoreCase.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

If IgnoreCase = True Then

Comp = vbTextCompare

Else

Comp = vbBinaryCompare

End If

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' This first large block of code determines whether the function

' is being called from a worksheet range or by another function.

' If it is being called from a worksheet, it must be called from

' a range with only one column or only one row. Two-dimensional

' ranges will cause a #REF error.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

If IsObject(Application.Caller) = True Then

If Application.Caller.Rows.Count > 1 And Application.Caller.Columns.Count > 1 Then

DistinctValues = CVErr(xlErrRef)

Exit Function

End If

''''''''''''''''''''''''''''''''''''''''''''''''''

' Save the size of the region from which the

' function was called and save a flag indicating

' whether we need to transpose the result upon

' returning.

''''''''''''''''''''''''''''''''''''''''''''''''''

If Application.Caller.Rows.Count > 1 Then

TransposeAtEnd = True

ReturnSize = Application.Caller.Rows.Count

Else

TransposeAtEnd = False

ReturnSize = Application.Caller.Columns.Count

End If

End If

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' Were we passed a Range object or a VBA array?

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

If IsObject(InputValues) = True Then

If TypeOf InputValues Is Excel.Range Then

''''''''''''''''''''''''''''''''''''''''''''''''

' Input is a Range object.

''''''''''''''''''''''''''''''''''''''''''''''''

If InputValues.Rows.Count > 1 And InputValues.Columns.Count > 1 Then

DistinctValues = CVErr(xlErrRef)

Exit Function

End If

If InputValues.Rows.Count > 1 Then

NumCells = InputValues.Rows.Count

Else

NumCells = InputValues.Columns.Count

End If

UB = NumCells

Else

DistinctValues = CVErr(xlErrRef)

Exit Function

End If

Else

'''''''''''''''''''''''''''''''''''''''''''''''''''''

' InputValues is not a Range object.

'''''''''''''''''''''''''''''''''''''''''''''''''''''

If IsArray(InputValues) = True Then

Select Case NumberOfArrayDimensions(InputValues)

Case 0

''''''''''''''''''''''''''''''''''''

' Zero dimensional array (scalar).

' Return an array of 1 element with

' that value.

''''''''''''''''''''''''''''''''''''

ReDim ResultArray(1 To 1)

ResultArray(1) = InputValues

DistinctValues = ResultArray

Exit Function

Case 1

UB = UBound(InputValues) - LBound(InputValues) + 1

'''''''''''''''''''''''''''''''''''''''''''''''''''''''

' If we were passed in an array from a worksheet

' function (e.g., =DISTINCTVALUES({1,2,3}), we

' need to set NumCells to the size of the input array.

' This is used later to properly resize the result array.

'''''''''''''''''''''''''''''''''''''''''''''''''''''''

If IsObject(InputValues) = False Then

NumCells = UB

End If

Case Else

DistinctValues = CVErr(xlErrValue)

Exit Function

End Select

Else

ReDim ResultArray(1 To 1)

ResultArray(1) = InputValues

DistinctValues = ResultArray

Exit Function

End If

End If

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' Ensure we don't have any NULLs or Objects in the InputValues.

' A Range object is allowed.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

For Each V In InputValues

If IsNull(V) = True Then

DistinctValues = CVErr(xlErrNull)

Exit Function

End If

If IsObject(V) = True Then

If Not TypeOf V Is Excel.Range Then

DistinctValues = CVErr(xlErrValue)

Exit Function

End If

End If

Next V

''''''''''''''''''''''''''''''''''''''''''''''''''

' Allocate the ResultArray and fill it with either

' vbNullStrings if we were called from a worksheet

' or with Empty values if called by a VB procedure.

'''''''''''''''''''''''''''''''''''''''''''''''''''

ReDim ResultArray(1 To UB)

For N = LBound(ResultArray) To UBound(ResultArray)

If IsObject(Application.Caller) = True Then

ResultArray(N) = vbNullString

Else

ResultArray(N) = Empty

End If

Next N

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' This is the logic that actually tests for duplicate values.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

ResultIndex = 1

''''''''''''''''''''''''''''''''''''

' We can always assume that the

' first element in the InputValues

' will be distinct so far.

''''''''''''''''''''''''''''''''''''

ResultArray(1) = InputValues(1)

''''''''''''''''''''''''''''''''''''''''

' Loop throught the entire InputValues

' array.

''''''''''''''''''''''''''''''''''''''''

For N = 2 To UB

'''''''''''''''''''''''''''''''''

' Set our Found flag = False. This

' flag is used to indicate whether

' we find Input(N) in the list of

' distinct elements. If we found it

' earlier, it is no longer a distinct

' element and we won't put it in the

' ResultArray.

''''''''''''''''''''''''''''''''''''

ElementFoundInResults = False

For M = 1 To N

'''''''''''''''''''''''''''''''''''''

' Scan through the array ResultArray

' looking for Input(N). If we find it,

' Input(N) is a duplicate so set the

' Found flag to True.

'''''''''''''''''''''''''''''''''''''

If StrComp(CStr(ResultArray(M)), CStr(InputValues(N)), Comp) = 0 Then

ElementFoundInResults = True

Exit For

End If

Next M

''''''''''''''''''''''''''''''''''''''''''''

' If we didn't find Input(N) in ResultArray

' then Input(N) is distinct so we increment

' ResultIndexand add Input(N) to ResultArray.

''''''''''''''''''''''''''''''''''''''''''''

If ElementFoundInResults = False Then

ResultIndex = ResultIndex + 1

ResultArray(ResultIndex) = InputValues(N)

End If

Next N

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' Here, we resize the ResultArray to the appropriate number of

' elements. ResultIndex is equal to the number of distinct elements found.

' If the function was called from a worksheet, ReturnSize is

' positive, equal to the number of cells in the array into which

' the function was entered and NumCells is the number of cells in

' the InputRange. If the function was called by another VB function,

' not from a worksheet, ReturnSizse and NumCells will be 0. Thus,

' if ReturnSize is not 0 and ResultIndex, the number of distinct elements,

' is less than the number of cells from in the InputValues, we

' set ResultIndex to the number of cells from which the function was called.

' This allows us in the For N loop that follows to pad out the

' entire Application.Caller range with vbNullStrings to prevent

' #N/A errors if the function is called from a range with more cells

' than there were disticnt elements. Note that this behavior differs

' from Excel's normal array formula handling.

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

If ReturnSize <> 0 Then

If ResultIndex < NumCells Then

If ResultIndex < ReturnSize Then

ResultIndex = ReturnSize

End If

End If

End If

ReDim Preserve ResultArray(1 To ResultIndex)

If UBound(ResultArray) > NumCells Then

For N = NumCells + 1 To ReturnSize

ResultArray(N) = vbNullString

Next N

End If

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' If we were called from a Column range on a worksheet (Rows.Count > 1),

' we need to transform ResultArray into a 2-dimensional array and transpose

' it so it will be properly stored in the column. Transpose1DArray does this

' function. If the function was not called from a worksheet, then the

' TransposeAtEnd flag will be false and we just return the array.

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

If TransposeAtEnd = True Then

DistinctValues = Transpose1DArray(Arr:=ResultArray, ToRow:=False)

Else

DistinctValues = ResultArray

End If

End Function

Function TransposeArray(Arr As Variant) As Variant

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' TransposeArray

' This function tranposes the array Arr. Arr must be

' a two dimensional array. If Arr is not an array, the

' result is just Arr itself. If Arr is a 1-dimensional

' array, the result is just Arr itself. If you need to

' transpose a 1-dimensional array from a row to a column

' in order to properly return it to a worksheet, use

' Transpose1DArray. If Arr has more than three dimensions,

' an error value is returned.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim R1 As Long

Dim R2 As Long

Dim C1 As Long

Dim C2 As Long

Dim LB1 As Long

Dim LB2 As Long

Dim UB1 As Long

Dim UB2 As Long

Dim Res() As Variant

Dim NumDims As Long

If IsArray(Arr) = False Then

TransposeArray = Arr

Exit Function

End If

NumDims = NumberOfArrayDimensions(Arr)

Select Case NumDims

Case 0

If IsObject(Arr) = True Then

Set TransposeArray = Arr

Else

TransposeArray = Arr

End If

Case 1

TransposeArray = Arr

Case 2

LB1 = LBound(Arr, 1)

UB1 = UBound(Arr, 1)

LB2 = LBound(Arr, 2)

UB2 = UBound(Arr, 2)

R2 = LB1

C2 = LB2

ReDim Res(LB2 To UB2, LB1 To UB1)

For R1 = LB1 To UB1

For C1 = LB2 To UB2

Res(C1, R1) = Arr(R1, C1)

C2 = C2 + 1

Next C1

R2 = R2 + 1

Next R1

TransposeArray = Res

Case Else

TransposeArray = CVErr(9)

End Select

End Function

Function NumberOfArrayDimensions(Arr As Variant) As Long

''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' NumberOfArrayDimensions

' This returns the number of dimensions of the array

' Arr. If Arr is not an array, the result is 0.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim LB As Long

Dim N As Long

On Error Resume Next

N = 1

Do Until Err.Number <> 0

LB = LBound(Arr, N)

N = N + 1

Loop

NumberOfArrayDimensions = N - 2

End Function

Function Transpose1DArray(Arr As Variant, ToRow As Boolean) As Variant

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' Transpose1DArray

' This function transforms a 1-dim array to a 2-dim array and

' transposes it. This is required when returning arrays back to

' worksheet cells. The ToRow parameter determines if the array is

' to be returned to the worksheet as a row (TRUE) or as a columns (FALSE).

' This should only be used for 1-dim arrays that are going back to

' a worksheet.

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Res As Variant

Dim N As Long

If IsArray(Arr) = False Then

Transpose1DArray = CVErr(xlErrValue)

Exit Function

End If

If NumberOfArrayDimensions(Arr) <> 1 Then

Transpose1DArray = CVErr(xlErrValue)

Exit Function

End If

If ToRow = True Then

ReDim Res(LBound(Arr) To LBound(Arr), LBound(Arr) To UBound(Arr))

For N = LBound(Res, 2) To UBound(Res, 2)

Res(LBound(Res), N) = Arr(N)

Next N

Else

ReDim Res(LBound(Arr) To UBound(Arr), LBound(Arr) To LBound(Arr))

For N = LBound(Res, 1) To UBound(Res, 1)

Res(N, LBound(Res)) = Arr(N)

Next N

End If

Transpose1DArray = Res

End Function