Arrays/Array functions

➤ Arrays are VBA variables which can store more than one item.

· the items held in an array are all of the same variable type

· one refers to an item by the array name and a number

```
syntax: declaration:   Dim Name(number)
            usage:          Name(x)        where   0 ≤ x ≤ number
```

· by default the indexing starts at 0

- Expl.: an array with three items named A

```
declaration:   Dim A(2)
usage:          A(0) = 5
                A(1) = 3
                A(2) = 6
note:          A(3) is not defined
```

• You may change the index set from its default value

```
syntax: declaration:   Dim Name(x to y)
            usage:          Name(z)        where   x ≤ z ≤ y
```

- Expl.: an array with three items named A

```
declaration:   Dim A(8 to 10)
usage:          A(8)   = 5
                A(9)   = 3
                A(10) = 6
note:          A(6), A(7), A(11), A(12), ... are not defined
```

• Alternatively you can also use the array function

```
syntax: declaration:   Dim Name as variant
           usage:          Name = array(x,y, ...,z)
```

· the indexing starts at zero, i.e. Name(0) = x

• Example 1:

```
Sub Example1()
    Dim A(8 To 10)
    A(8) = 2
    A(9) = 3
    A(10) = A(8) + A(9)
    Range("A10").Value = A(10)
End Sub
```

- writes 5 into the cell A10 of the active worksheet

• Example 2:

```
Sub Example2()
    Dim B As Variant
    B = Array(2, 3, 4, 5)
    Range("A13").Value = (B(0) + B(1)) /B(3)
End Sub
```

- writes 1 into the cell A13 of the active worksheet

➤ Multidimensional arrays are VBA variables which can hold more than one item related to several index sets (up to 60)

· e.g. a two dimensional array is a matrix

syntax: declaration:	Dim Name(num1,num2,num3,...)	
usage:	Name(x,y,z,...)	$0 \le x \le$ num1
		$0 \le y \le$ num2
		$0 \le z \le$ num3
		....

· the change of the index set is analogue to the one dimensional case

- Expl.: a 2 by 2 matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$

declaration:	Dim A(1 to 2,1 to 2)		
usage:	A(1,1) = a	A(1,2) = b	
	A(2,1) = c	A(2,2) = d	

➤ Resizable arrays are arrays whose size is not fixed

```
syntax: declaration:   Redim Name(x to y)
                       ........
                       Redim Name(w to z)
```

· the first statement creates a one dimensional resizable array

· the second statement overwrites the first statement

```
syntax: declaration: Redim Name(x to y)
                        ........
                     Redim preserve Name(w to z)    w≤x , z≥y
```

· now the values in the array Name(x to y) will be saved

➤ Upper and lower bound function

· Lbound(RA) gives the lower bound of the array called RA

· Ubound(RA) gives the upper bound of the array called RA

```
- Expl.:  Redim RA(1 to 10)
          x = Lbound(RA)            ( x =  1)
          y = Ubound(RA)            ( y = 10)
          Redim RA(12 to 19)
          x = Lbound(RA)            (now  x =  12)
          y = Ubound(RA)            (now  y = 19)
```

➤ Data exchange: Arrays can be used as an efficient way to exchange data between the Excel spreadsheet and the VBA program

• VBA program → spreadsheet

```
Range("A1:B2").Value = A
```

(puts the values of the array A into cells A1:B2)

• spreadsheet → VBA program

```
Dim B As Variant
B = Range("A1:B2").Value
```

(assigns the values of cells A1:B2 to the array B)

- Expl.: The content of two 2 by 2 matrices in the cells A1:B2 and D1:E2 are read to two arrays A and B. The matrices are multiplied and the result is returned to the cells G1:H2.

```
Sub Matrix()
    Dim A, B As Variant            ⇐ arrays have to be variants
    Dim C(1 To 2, 1 To 2)
    A = Range("A1:B2").Value
    B = Range("D1:E2").Value
    For i = 1 To 2                 ⇐ the indexing starts at 1
        For j = 1 To 2
            C(i, j) = A(i, 1) * B(1, j) + A(i, 2) * B(2, j)
        Next j
    Next i
    Range("G1:H2").Value = C
End Sub
```

➤ <u>MMULT</u> is an Excel array function which returns the product of two arrays

syntax: MMULT(array name1 , array name2)

- Expl.: MMULT(“A1:B2“ , “D1:E2“)

  ⇒ returns the same product as the previous VBA program

- notice that MMULT is an array function, such that you have to prepare for an output bigger than one cell: (recall LINEST)
  · select a range for the output, e.g. 2×2 cells
  · type the function, e.g. = MMULT(.....)
  · complete with Ctrl + Shift + Enter

- notice also: MMULT is an Excel function not VBA function

➤ <u>The Split Function</u> returns an array consisting of substrings from a string expression in which each substring is separated by a delimiter which can be specified

syntax: Split(expression [, delimiter] [, limit])

expression ≡ a string expression

delimiter ≡ the character which separates the substrings (the default value is space)

limit ≡ the maximum number of substrings to be returned (the default value is –1, that is all substrings)

- Expl.:
```
Dim x as variant
x = Split("Today is Tuesday")
 ⇒ x(1) = "Today"  x(2) = "is"  x(3) = "Tuesday"
```
or:
```
x = Split("a,b,c,d,e,f,g" ,  "," , 3)
 ⇒ x(1) = "a"  x(2) = "b"  x(3) = "c,d,e,f,g"
```

➤ <u>The Join Function</u> returns a string consisting of the values in a string array separated by a specified delimiter

syntax: Join(sourcearray [, delimiter])

sourcearray ≡ an array containing strings

delimiter ≡ the character which separates the substrings (the default value is space)

- Expl.:
```
Dim x(1 to 3)
 x(1) = "Today"
 x(2) = "is"
 x(3) = "Tuesday"
 y = Join(x)
  ⇒ y = "Today is Tuesday"
```

- similarly:

    y = “Today “ & “is “ & “Tuesday“

    ⇒ y = “Today is Tuesday“

· in addition:

```
Dim x as integer
x = 8
y = “Today “  & “is “  & “Tuesday the “ & x & “-th of March“
```

    ⇒ y = “Today is Tuesday the 8-th of March“

· here the individual components do not have to be of string type (8 is an integer)