School of Business
and Social Sciences

EFMD EQUIS ACCREDITED

VBA introduction course
Part 2

For

Financial Engineering // MSc Finance

By the Analytics Group

See more at www.asb.dk/ag

Analytics Group

From part 1

From the VBA course part 1, you should be familiar with

- What is VBA?
- The Macro recorder
- The VBA environment
- Creating a procedure
- Debugging
- Variables and datatypes
- Functions
- Navigation in VBA
- Commenting

Agenda for today

Today we will cover the following topics

- Controlling program flow:
    - If...Then...Else...Endif
    - Loops:
        - For...Next
        - Do While/Until or While-Wend
- Arrays
- Set Range
- Briefly see examples of a message box.

All the topics will be covered with examples and small problems

Remember how to create a procedure

In the VBA editor click on **Insert → Procedure** and choose **Sub** or **Function**, depending on which procedure you wish to work in. You can also write the commands manually

Syntax:

```
Sub name()
Dim var1 as type.......
...code...
End Sub

Function name(var1 as type, var2 as type, ....) as type
...code...
End Function
```

Remember to Declare variable types

Remember to ‘Dim’ variable types, i.e.

- ‘as Double’ (Numeric)
- ‘as Integer’ (Rounded number)
- ‘as String’ (Text)

If-sentences

Used when there are certain conditions which must be met when choosing between two or more different options.

There are many ways to formulate the if-sentences.

The simplest way to code the condition is by using the ”One-Line If Statement”.

```
Function Functionname(var1, var2...)
          If ..statement.. Then ..do someting.. Else ...do something else...
end function
```

Ex.:

```
Function Vbafunction(x)
          If x > 1 Then vbafunction = 1 Else vbafunction = x
end function
```

If-sentences

If-sentences can also be applied by using multiple lines. The code is then given by.

```
Function Functionname(var1, var2...)
        If Value > 0 Then
                strike = Value
        Else
                strike = 0
        End If
end function
```

The code can also be extended as an "If – ElseIf – Statement", which is shown in the table below below.

Simple	Else	Elseif
If condition *then* *...code...* *End if*	*If* condition *then* *...code...* *Else* *...code...* *End if*	*If* condition *then* *...code...* *Elseif* condition *then* *...code...* *Else* *...code...* *End if*

If-sentences

The if-conditions can be extended even further by writing:

```
If Condition1 Then
        Statement...
ElseIf Condition2    Then
        Statement ...
ElseIf Condition3    Then
        Statement ...
   [...More ElseIfs...]
Else
        Statement ...
End If
```

Useful operators

Operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
=	Equal to
<>	Not equal to/different from
Or	Or
And	And

Example 1

We should code a function named *example1*, which will provide the rate of a loan, based on the actual rate.

If the actual rate is below 1%, then the rate of the loan should be 1%.

If the actual rate is between 1% and 4% the rate of the loan should be the actual rate.

If the actual rate is above 4%, then the rate of the loan should be 4%.

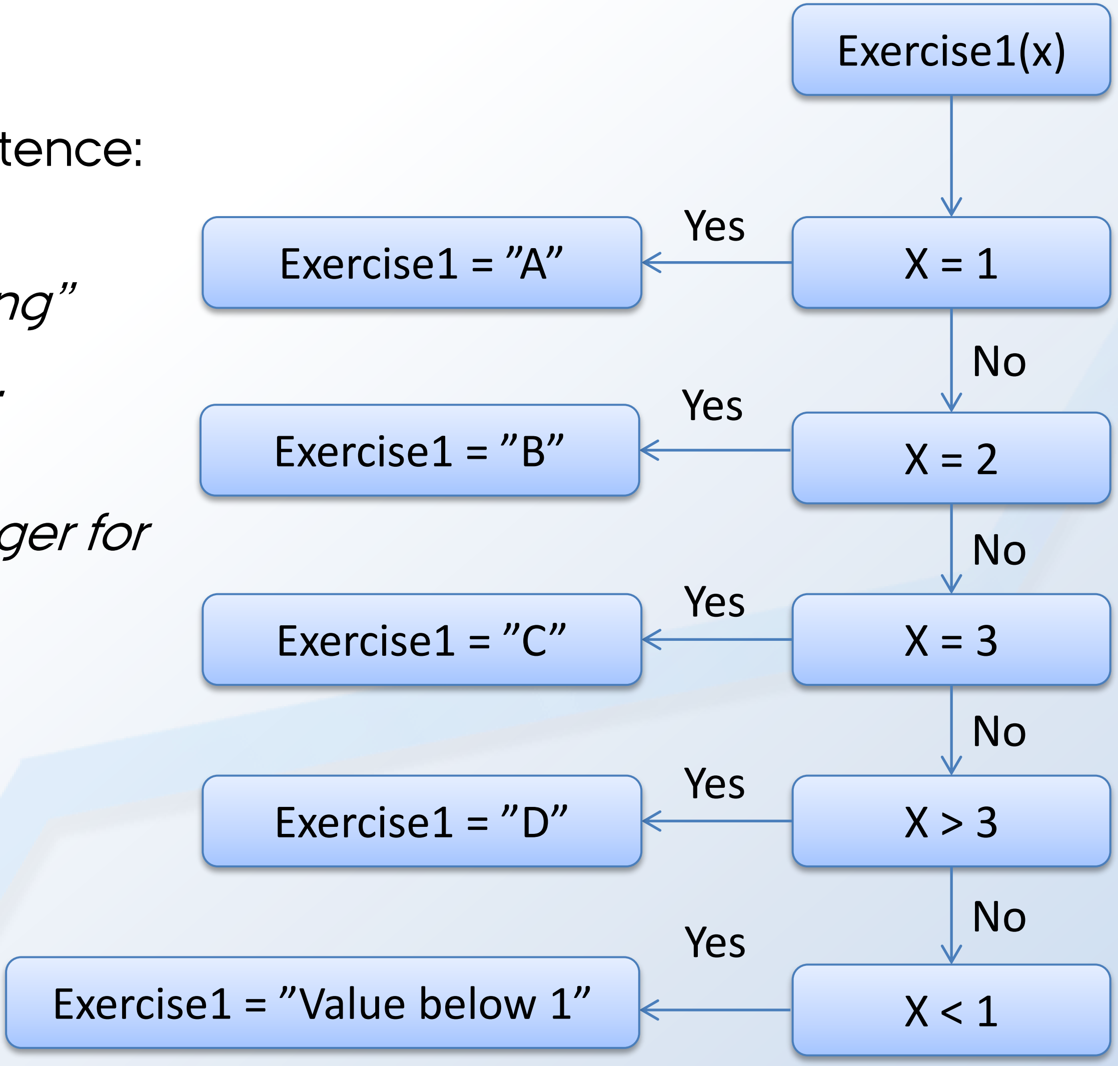
Actual rate	Rate of loan
] - ∞; 1%[	1%
[1% - 4 %]	Actual rate
[4% - ∞]	4%

Exercise 1

Code the following if-sentence:

Use the variabletype "string" for the function exercise1.

Use the variabletype integer for the variable x.



Exercise1(x)

X = 1
Yes
Exercise1 = "A"
No

X = 2
Yes
Exercise1 = "B"
No

X = 3
Yes
Exercise1 = "C"
No

X > 3
Yes
Exercise1 = "D"
No

X < 1
Yes
Exercise1 = "Value below 1"

Exercise 2

You are to code a function, which will determine the annual coupon of a bond. The annual coupon is calculated as:

C = F x R (F = Facevalue, R = Rate)

The rate is determined by the classification of the bond. The classification table to the right shows the coupon rates.

- After you have coded the function, assign a help text to the function, which will help you when you locate your function through the **Insert Function Wizard**

Bond type	Rate
AAA	3,0%
AA	4,0%
A	5,0%
B	6,0%
else	8,0%

Hint: You *could* (but don't have to) use the function "Ucase(bond)" in order for Excel to be indifferent between capital letters and small letters. (e.g. "AAA" and "aaa").

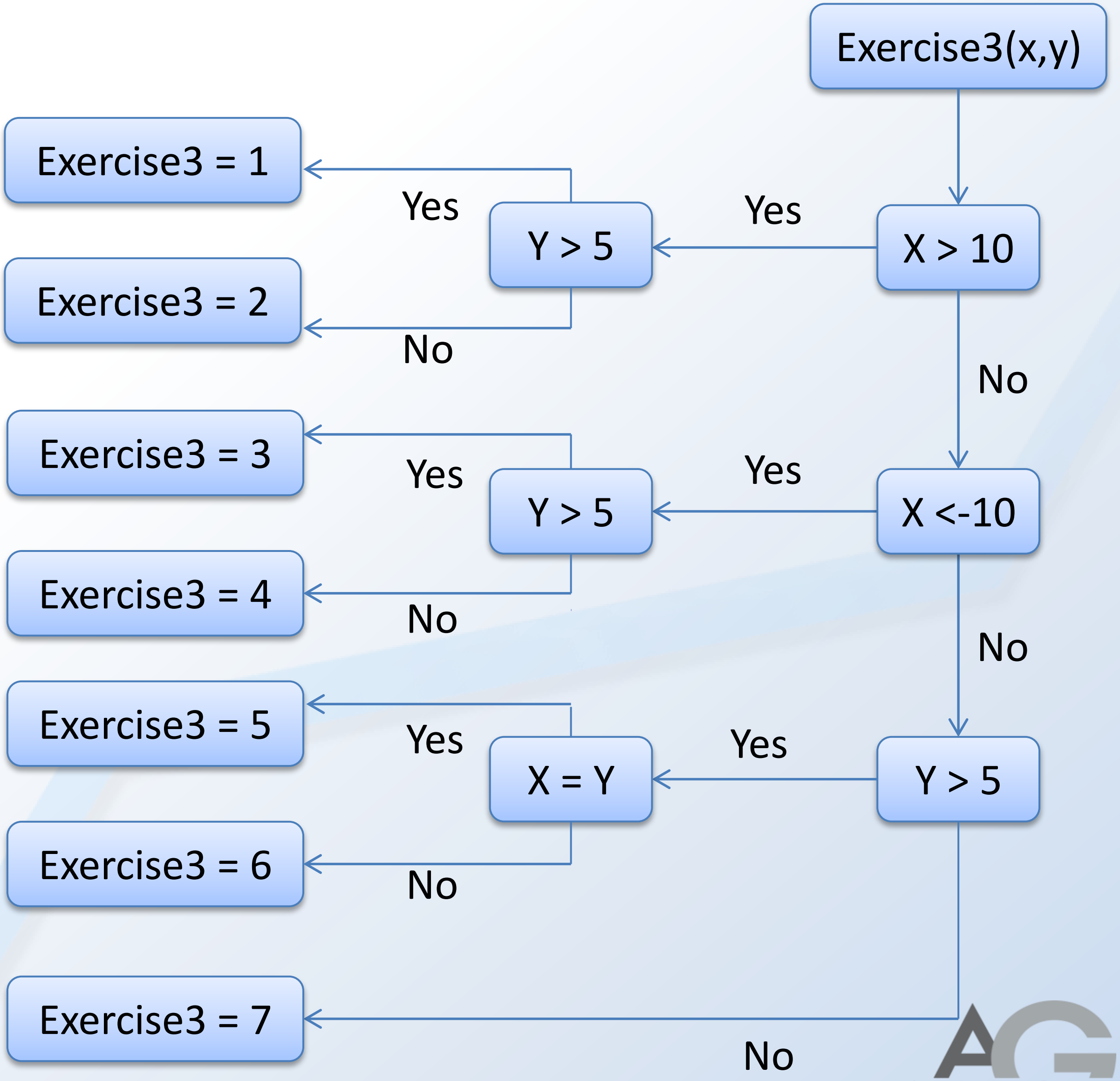
Exercise 3 – do at home

Code the following function in Excel.

Be aware, that there now is 2 variables, instead of 1.

Hint. Use if -statements within a if-statement in order to make this code correct.

Source – Benninga!



Exercise3(x,y)

X > 10
Yes → Y > 5
Yes → Exercise3 = 1
No → Exercise3 = 2
No → X <-10
Yes → Y > 5
Yes → Exercise3 = 3
No → Exercise3 = 4
No → Y > 5
Yes → X = Y
Yes → Exercise3 = 5
No → Exercise3 = 6
No → Exercise3 = 7

Loops - Iterations

For...Next

Loops are used to run the same piece of code repeatedly. For those of you used to SAS coding, this should be familiar, as loops are often used when coding in SAS.

The For...Next loop is found in three versions:

Simple loop	Simple loop with step	Simple loop with reverse step
For i = 1 To 10 …code… *Next i*	*For i = 0 To 30 step 3* ...code... *Next i*	*For i = 10 To 0 step -2* ...code... *Next i*

Example 2

We would like to write a function, which calculates the present value of a given cash flow for 5 periods.

The formula should be well known and is defined as:

$$NewPV(CF,r) = \frac{CF}{(1+r)^1} + \frac{CF}{(1+r)^2} + \frac{CF}{(1+r)^3} + \frac{CF}{(1+r)^4} + \frac{CF}{(1+r)^5}$$

We assume a fixed rate and a fixed cash flow throughout the 5 year period.

Source: Benninga, exercise 37,3,

Exercise 4

You should write a function, which calculates the present value of a given cash flow for n periods.

The formula is once again the same, however now we have the number of periods as another variable in the code.

$$Exercise5(CF, r, n) = \sum_{i=1}^{n} \frac{CF}{(1+r)^i}$$

We assume a fixed rate and a fixed cash flow

Check your results

CF: Cash Flow = 6
r: Rate = 5%
n: Number of Years = 5

PV: Present Value = 25.98

Loops - Iterations

Do…while/until…loop

We don’t always know how many times the piece of code must be run. If this is the case we can use the do loop.

Two versions:

-Do While: repeat code as long as certain conditions are met.

-Do until: repeat code until a certain condition is met.

The Do loop syntax is:

```
Do [While/Until] condition
…code…
Loop
```

While…Wend operation can also be used.

Note! Make sure the loop has an end. Otherwise we have an infinite loop [ESC].

Example 3

We want to code a function, which provide information about the number of years until a certain amount of investment is met.

Each year, we should deposit an amount of money. This amount is fixed for all years.

We receive interest on the accumulated amount of investment each year.

The function should have 3 variables:

- Total investment required
- Deposit invested each year
- Rate of return on the investment

Example 3 - Extended

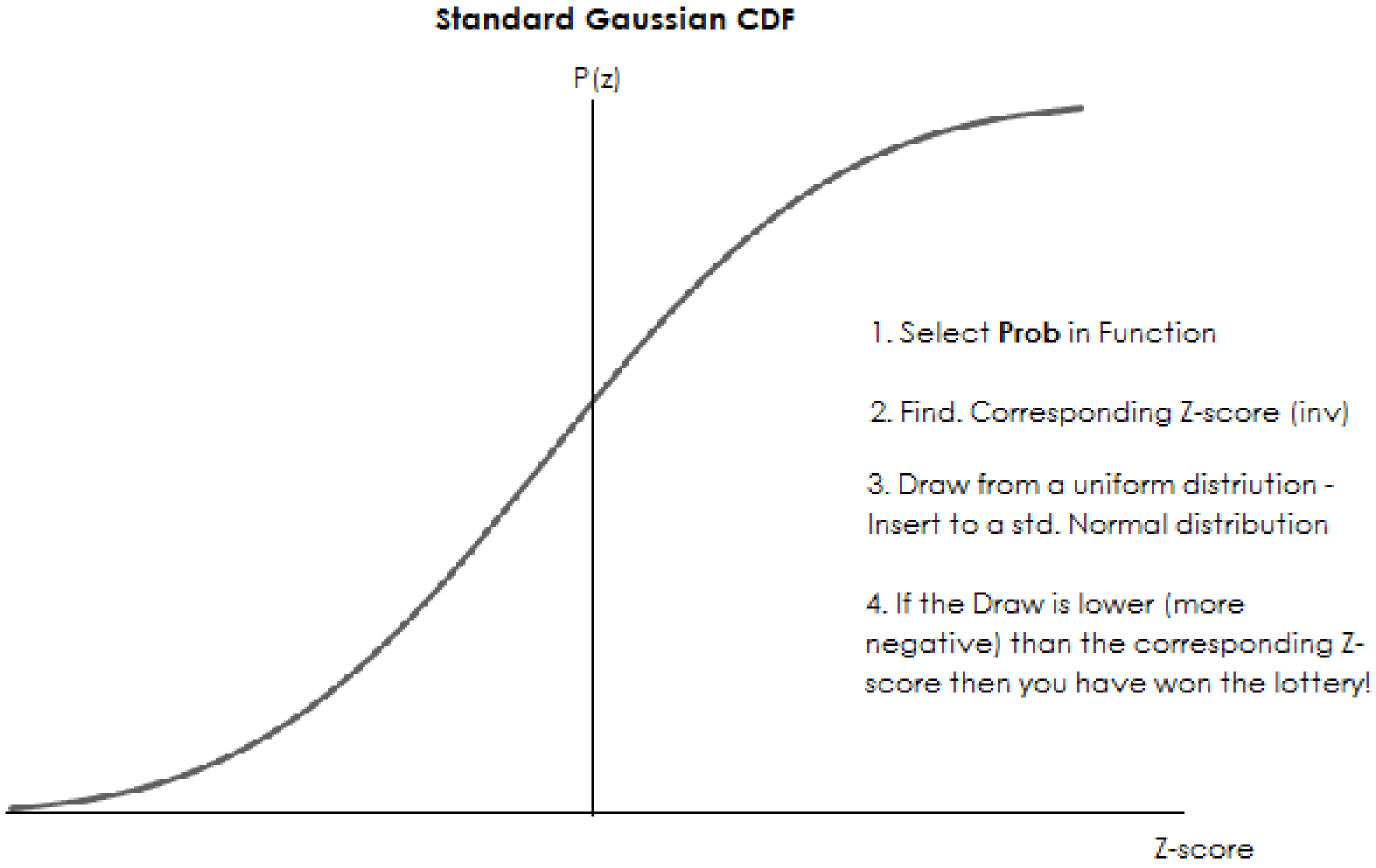
We now want to extend the previous example by introducing uncertainty.

Each year, the person will bet on the lottery – if he/she wins the big score the total investment is assumed to be met.

Excel can only generate from a uniform distribution [0;1].

We want to generate from a normal distribution (even though it does not make sense in this particular case).

Example 3 - Extended



1. Select **Prob** in Function

2. Find. Corresponding Z-score (inv)

3. Draw from a uniform distriution - Insert to a std. Normal distribution

4. If the Draw is lower (more negative) than the corresponding Z-score then you have won the lottery!

The transformation to the gaussian distribution is of no use what so ever in this example, However, it has been added as an example to show how to draw from another distr. than the uniform distribution

/IAM

Exercise 5

Create a function for a loan. You should incorporate an interest rate and a payment, which must be payed annually. The function should have 3 variables:

- Initial loan amount
- Interest rate
- Annual payment

Ex:
A man buys a car today at a price of $30,000 and he borrows at an interest rate of 10%.

He pays $5,000 up front.

How many years will he have to repay $5,000 before he has paid of his entire loan?

Exercise 6 – do at home

From CFII you should be familiar with the formula of Black & Scholes

Create a function that returns the Black-Scholes value for put and call options (Note the model is including dividends as Merton (1973)).

The black-scholes formula is given by

$$call = S * N(d_1) * e^{-qT} - X * e^{-rT} N(d_2)$$

$$put = X * e^{-rT} N(-d_2) - S * N(-d_1) * e^{-qT}$$

$$d1 = \frac{\ln(S/X) + (r - q + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d2 = d1 - \sigma\sqrt{T}$$

Input variables:

Spot (S), Strike (X), Time to maturity (T), Volatility (v), Risk free interest rate (r), dividend yield (q), CallPut (o)

N() = cumulative normal distribution function

Exercise 6 - results

You can check if your formula is valid. The result for a put and call option with the following input should yield

Spot, S: 100
Strike, X: 90
Volatility, V: 40 %
Risk free rate, r: 10 %
Dividend yield, q: 5 %
Time to maturity, T: 1

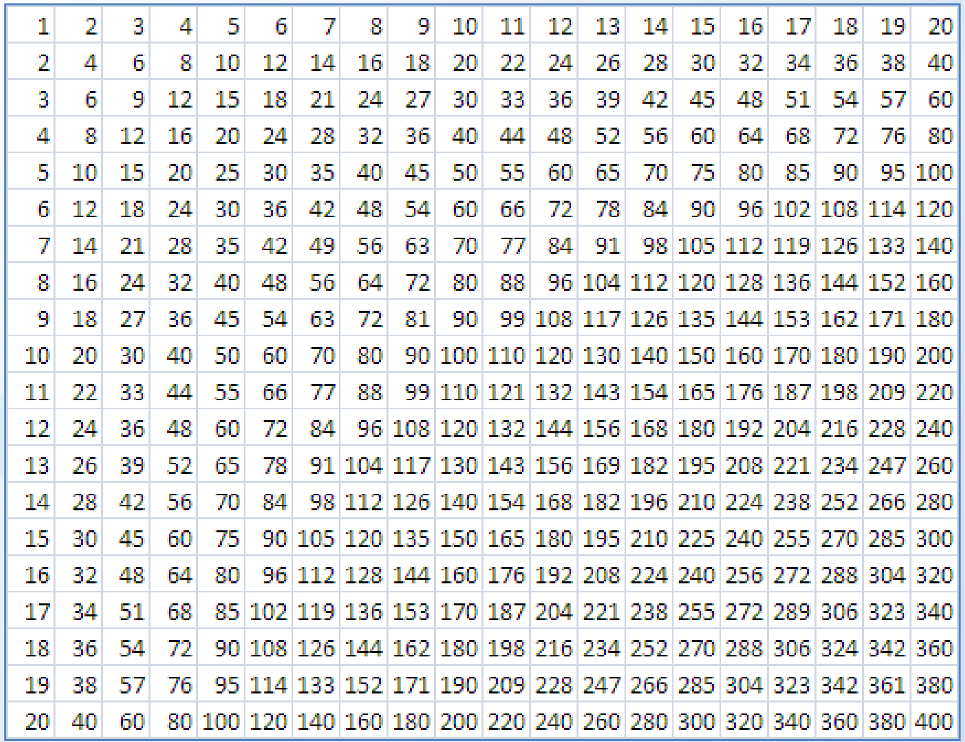
Put value: 8,1762
Call value: 21,8638

Example 4

This exercise combines navigation in Excel with loops.

Make a subroutine that can create a 20x20 table like the one shown to the right.

Hint: use the VBA function cells(i, j)



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80
5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108	114	120
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	152	160
9	18	27	36	45	54	63	72	81	90	99	108	117	126	135	144	153	162	171	180
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	176	187	198	209	220
12	24	36	48	60	72	84	96	108	120	132	144	156	168	180	192	204	216	228	240
13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	208	221	234	247	260
14	28	42	56	70	84	98	112	126	140	154	168	182	196	210	224	238	252	266	280
15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255	270	285	300
16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	256	272	288	304	320
17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	272	289	306	323	340
18	36	54	72	90	108	126	144	162	180	198	216	234	252	270	288	306	324	342	360
19	38	57	76	95	114	133	152	171	190	209	228	247	266	285	304	323	342	361	380
20	40	60	80	100	120	140	160	180	200	220	240	260	280	300	320	340	360	380	400

Sum all the numbers into a single cell.

Hint: use the worksheetfunction ”sum”…!

The number should be **44,100**

Arrays

Anytime you need to loop through the same code to grab different variables, you should consider using an array to store those variables. This will often reduce the amount of code, and usually make the code more efficient than non-array code.

For instance if you need to store a numeric value for each day of the year. You could declare 365 separate numeric variables which is a lot of work and makes your code run slower. Instead you should create an array to store all the data in one variable. The array itself is a single variable with multiple elements, where each element can contain one piece of data.

You can add as many dimensions to your array as you like, but beware, the more dimensions your arrays get, the more complex they become to manage.

Arrays can be used in subs as well as in functions.

Arrays

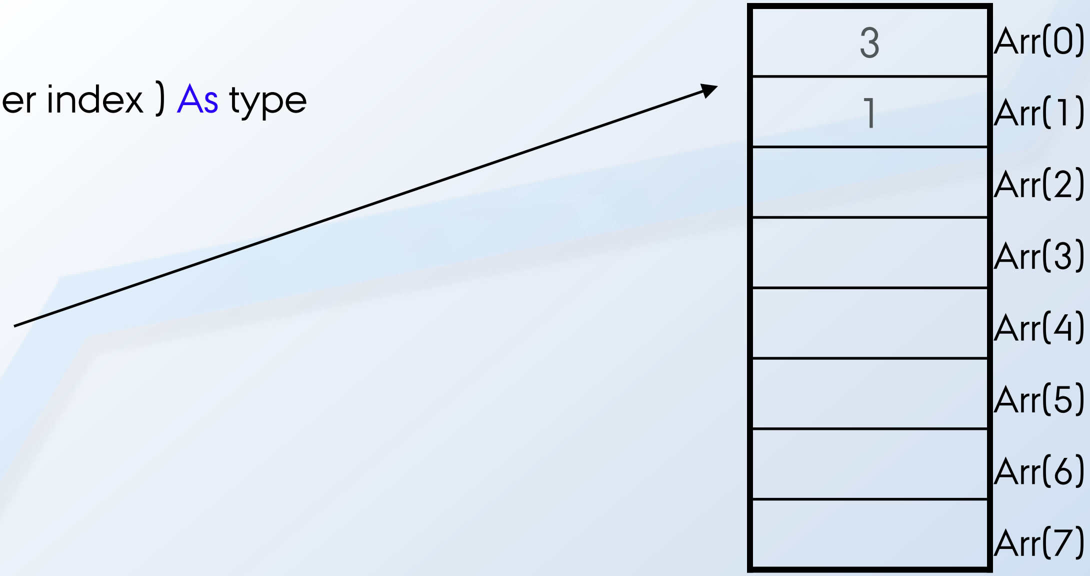
Arrays can be compared to columns in Excel. They are just invisible to the user and exist in the memory of the computer. When declaring an array, we must decide how many spaces to include. Spaces can be compared to the number of rows in a column.

Note: Array's start by default at index nr=0, so if we want 3 places, only set number of rows to 2.

A standard array is initialized as

```
Dim arrayname( upper index ) As type
```

```
Dim Arr(7)
Arr(0) = 3
Arr(1) = 1
```



3	Arr(0)
1	Arr(1)
	Arr(2)
	Arr(3)
	Arr(4)
	Arr(5)
	Arr(6)
	Arr(7)

Two dimensional array

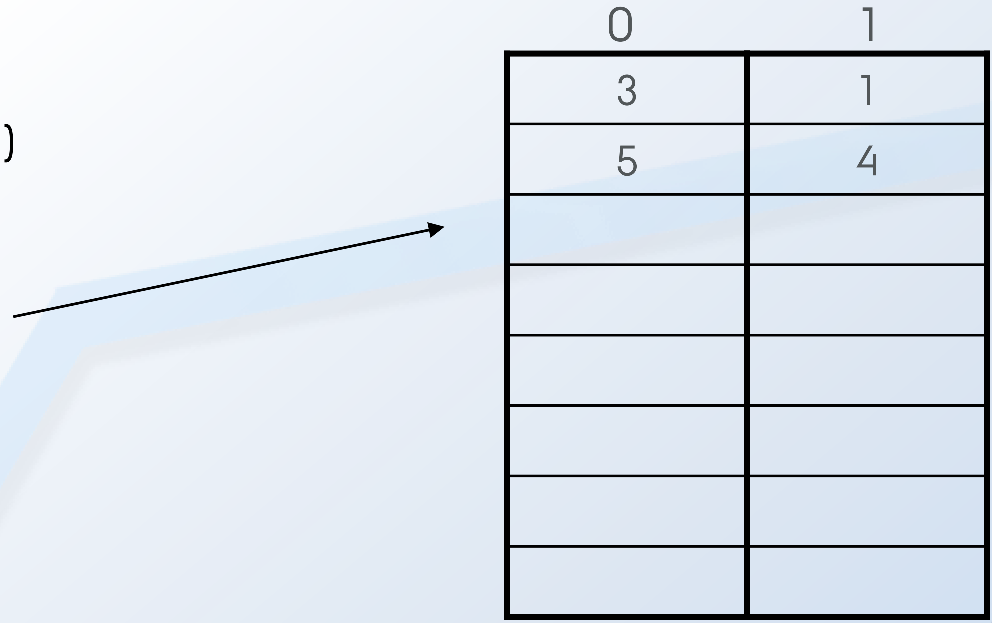
A two dimensional array can again be compared to an Excel worksheet. Now we just have the option of including multiple columns.

Dim arrayname(~~upper column index, upper row index~~) As type

(upper ROW index, upper COLUMN index)

Dim Arr(7,1)

Arr(0,0) = 3
Arr(1,0) = 5
Arr(0,1) = 1
Arr(1,1) = 4



0	1
3	1
5	4

Arrays

The dimensions of an array can also be set with exact numbering. i.e. if we want an array with the indexed rows 5-15:

```
Dim arr(5 to 15)
```

Equivalently for two dimensions with rows 5-15 and columns 10-20:

```
Dim arr(5 to 15, 10 to 20)
```

We can also declare a dynamic array, by leaving the dimensions empty.

```
Dim arrayname( ) As type
```

The array can be resized as the program is running

```
ReDim arrayname( index ) As type
```

Remember to use index numbers when working with arrays

Arrays

Remark

You can use the **Option Base** statement to specify the first index number of an array. So by writing

Option Base 1

at the beginning of the module, you will obtain that all your arrays will start with index 1, i.e. the statement

```
Dim arr(7) as Type
```

Is equivalent to the statement

```
Dim arr(1 to 7) as Type
```

The default is set to 0.

Example 5

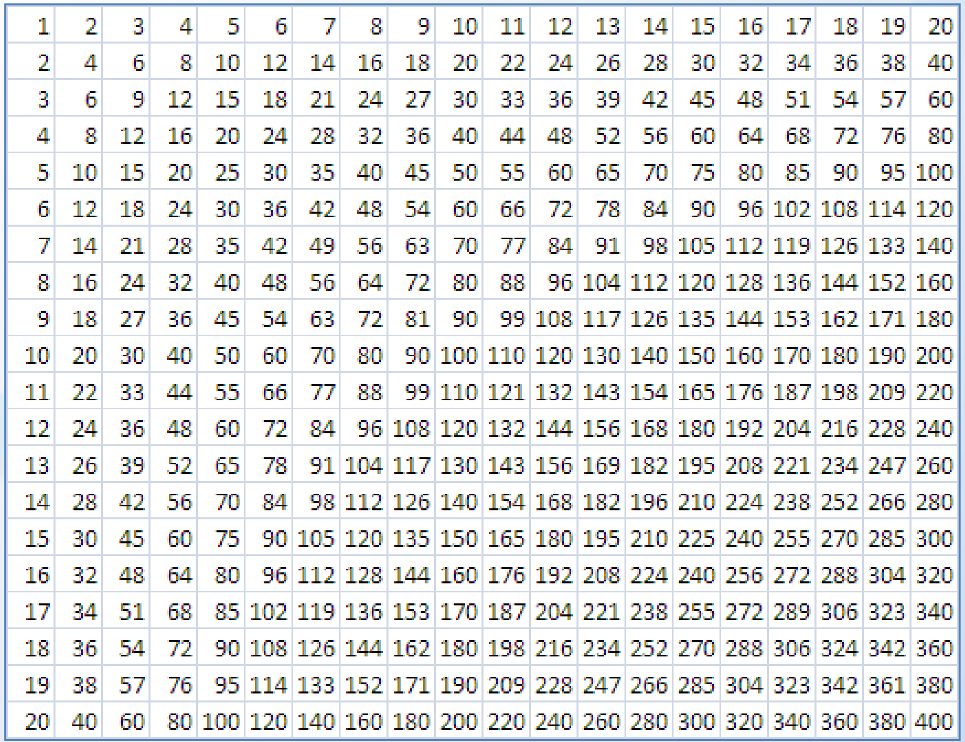
In this example we want to create a subroutine which provides the numbers from 1 to 100 and store them in an array.

Finally we want to output the content of the array to our spreadsheet.

Exercise 7

This is Example 4 but solved using arrays.

Make a subroutine that can create a 20x20 table like the one shown to the right.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80
5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108	114	120
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	152	160
9	18	27	36	45	54	63	72	81	90	99	108	117	126	135	144	153	162	171	180
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	176	187	198	209	220
12	24	36	48	60	72	84	96	108	120	132	144	156	168	180	192	204	216	228	240
13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	208	221	234	247	260
14	28	42	56	70	84	98	112	126	140	154	168	182	196	210	224	238	252	266	280
15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255	270	285	300
16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	256	272	288	304	320
17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	272	289	306	323	340
18	36	54	72	90	108	126	144	162	180	198	216	234	252	270	288	306	324	342	360
19	38	57	76	95	114	133	152	171	190	209	228	247	266	285	304	323	342	361	380
20	40	60	80	100	120	140	160	180	200	220	240	260	280	300	320	340	360	380	400

Hint: You have to define your array as two Dimensional

Sum all the numbers into a single cell.

Hint: use the worksheetfunction "sum"...!

The number should be **44,100**

Using loops to read variables into arrays

You saw before how to store values in an array and paste it to your spreadsheet. Now we will show how to read a range from the spreadsheet into an array and use the data within the array to do calculations.

Whenever you read data from a range into an array the type of the array HAS to be of type variant.

```
Dim arrayname() As Variant
```

When you read changing/dynamic ranges into your array within a loop the syntax gets a little more complicated. An example:

```
For i = 1 To 100
        arrayname = Range("a" & i & ":a" & i + 10).Value
Next i
```

This results in a array with 10 data points.

Example 6

- Open the spreadsheet **Reading numbers.xlsx**

- The sheet contains 36 data points which we want to calculate a smoothing average for. The smoothing average should be calculated as a simple average using 3 data points at a time.

Set Range

When working with subroutines it might come in handy to refer to a range as:

Set *range =* Worksheets(“*worksheetname*").Range(“*XX*", “ *YY*")

Example:

```
Set optiondata = Worksheets("Option_Data").Range("A12", "C193")
```

Then we can refer to the range *optiondata* later in the code instead of refering to specific cells.

Example 7

- Open the spreadsheet **Option Data.xlsx**

- The spreadsheet contains put and call prices for a number of options with different strikes, but with fixed time to maturity and interest rate.
- All options are written on the same underlying stock, so the spot is also the same for all options
- No dividends are assumed (q=0).

Task: Create a routine that will look through all call options and see if the upper bound is violated – if a violation is found, a message box should address the location of the violation.

Upper bound violation (call): $C > S(0)$

Exercise 8

- Use the spreadsheet **Option Data.xlsx**
- Create a subroutine that will look through the put prices and locate the first upper bound violation (if any).
- If a violation is found – a message box should address the location
- If no violation is found – a message box should indicate this.

Upper bound violation (put): $P > K * \exp(-T * r)$

(*K=strike)*

Advanced exercise 9 – do at home

- Open the Excel sheet **Beta Assignment.xlsx**
- We want to calculate the rolling beta for a share(**Y**) using S&P500(**X**) index as proxy for the market index.
- Rolling beta is calculated as ordinary beta using a regression. However, here we roll the observations over for each regression. The first beta is calculated using data from observations 1:60. The second for using 2:61 and so on.
- You should end up with 300 beta estimates.
- Hint:Use application.worksheetfunction.Slope(**Y**;**X**) to calculate each beta

Recommended readings

For coding in the course 'Financial Engineering' it might be a helpful to read the following chapters in Benninga, Simon – Financial Modeling:

- Chapter 36 – User-Defined functions with VBA
- Chapter 37 – Types and Loops
- Chapter 39 – Arrays

If you wan't to do some exercises, then it is recommended that you solve:

- Chapter 36: Ex: 1,2,3,4,5,6
- Chapter 37: Ex: 3,4,5,6,7,8

End of VBA courses

Thoughts to live by as a programmer!

- Programming is about being lazy!
- Nothing is impossible, when it comes to programming
- Programming is learned by trial-and-error

Remember to evaluate the course on:

au.dk/it → For Students → IT at School of Business and Social Sciences → VBA → Course Evaluation