Variable Types, IF structures and Flow charts

We saw last week that in VBA there are seven different types of variables: Integer, Single, Double, String, Boolean, Variant and Date. Today we will see in some more detail what they are and what their values can be.

- A bit is a basic unit of information storage and communication (is short for binary digit).
- Integer type variables are stored in memory as 16-bit (2-byte) numbers ranging in value from -32768 to 32767.
- String can contain up to approximately 2 billion (2^31) (letters & symbols) characters.
- Date are stored as 64-bit (8-byte) numbers that represent dates ranging from 1 January 100 to 31 December 9999 and times from 0:00:00 to 23:59:59.

- Boolean are stored as 16-bit (2-byte) numbers, but they can only be True or False.
- Single (single precision) variables are stored as 32-bit (4-byte) numbers, ranging in value from -3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values.
- Double (double precision) variables are stored as 64-bit (8-byte) numbers ranging in from -1.7976931348623 1E308 to -4.940656458412 47E-324 for negative values and from 4.940656458412 47E-324 to 1.7976931348623 2E308 for positive values.
- Variant is the data type for all variables that are not explicitly declared as some other type.
- The more storage space (bits) a variable needs, the slower a program using it will be!

Working with dates and times in VBA

VBA handles dates by associating each date to a number!

1-st of January 100 = -657434

……

1-st of January 1900 = 1

2-nd of January 1900 = 2

……..

12-th of November 2009 = 40129

Because VBA associates dates to numbers, it makes sense inside a VBA program to “subtract” two dates. If we do that we will obtain the number of days that have passed between the earlier and the later date!

There are several useful VBA-functions that take variables of date type as input. You can use these functions inside VBA programs:

- Month(date) → a number between 1 and 12 representing the month
- Weekday(date) → a number between 1 and 7 representing the day with Sunday=1, Monday=2, …, Saturday=7

- Year(date) → a number between 100 and 9999 for the year
- Hour(date) → a number between 0 and 23 for the hour
- Minute(date) → a number between 0 and 59 for the minute
- Second(date) → a number between 0 and 59 for the second

As in Excel, the function Now() returns the current date and time.

Examples:

a) Write a UDF which computes the weekday for a date

```
Function DD(da As Date) As Integer
        DD = Weekday(da)
End Function
```

· Format the cell A1 as date and enter 25/10/2005

· "=DD(A1)" → 3

b) Write a UDF which calculates the age in years given the birthdate.

```
Function age(birthdate As Date) As Integer
                age = Int((Now() - birthdate) / 365)
End Function
```

· (Now() - birthdate) ≡ the age in days

· Int(x) ≡ extracts the integer part of x

· age ≡ the age in integer numbers of years

Declaration of constants

- Constants are variables which do not change their value during the execution of the program (UDF).
- Constants are used to keep the programming structure clear and to avoid tedious re-typing or time consuming re-calculations.
- You can declare constants

  i) such that they are only available inside the program or

  ii) such that they are available in the entire worksheet.

Syntax: i) **Const** name [**As** type] = value

ii) **Public Const** name [**As** type] = value

Function .....

It is important to do the **Public Const** statement before the **Function** statement.

- Expl. a) Const Pihalf = 1.570796327

  b) Const Errmess as string = “Division by zero!!!”

  c) Public Const Errmess as string = “Division by zero!!!”

Declaration of variables

When writing VBA programs it is often useful to define variables that are only available inside the program. If you are dealing with long codes, this can make the writing tidier and the code easier to understand.

In order to define such variables you have to write:

Dim Variable Name [**As** Type]

Inside the programs code, before the variable is used.

For example the function:

```
Function  exa(x As Single) As Single
      Exa= Exp(x)+ Exp(2*x)+Exp(3*x)+Exp(4*x)
End Function
```

Could also be written as:

```
Function  exa(x As Single) As Single
Dim y As Single
      y=exp(x)
      Exa= y+ y^2+y^3+y^4
End Function
```

- Writing any kind of computing program consists of three basic principal steps:

  i) Design an algorithm which will perform the task you want.

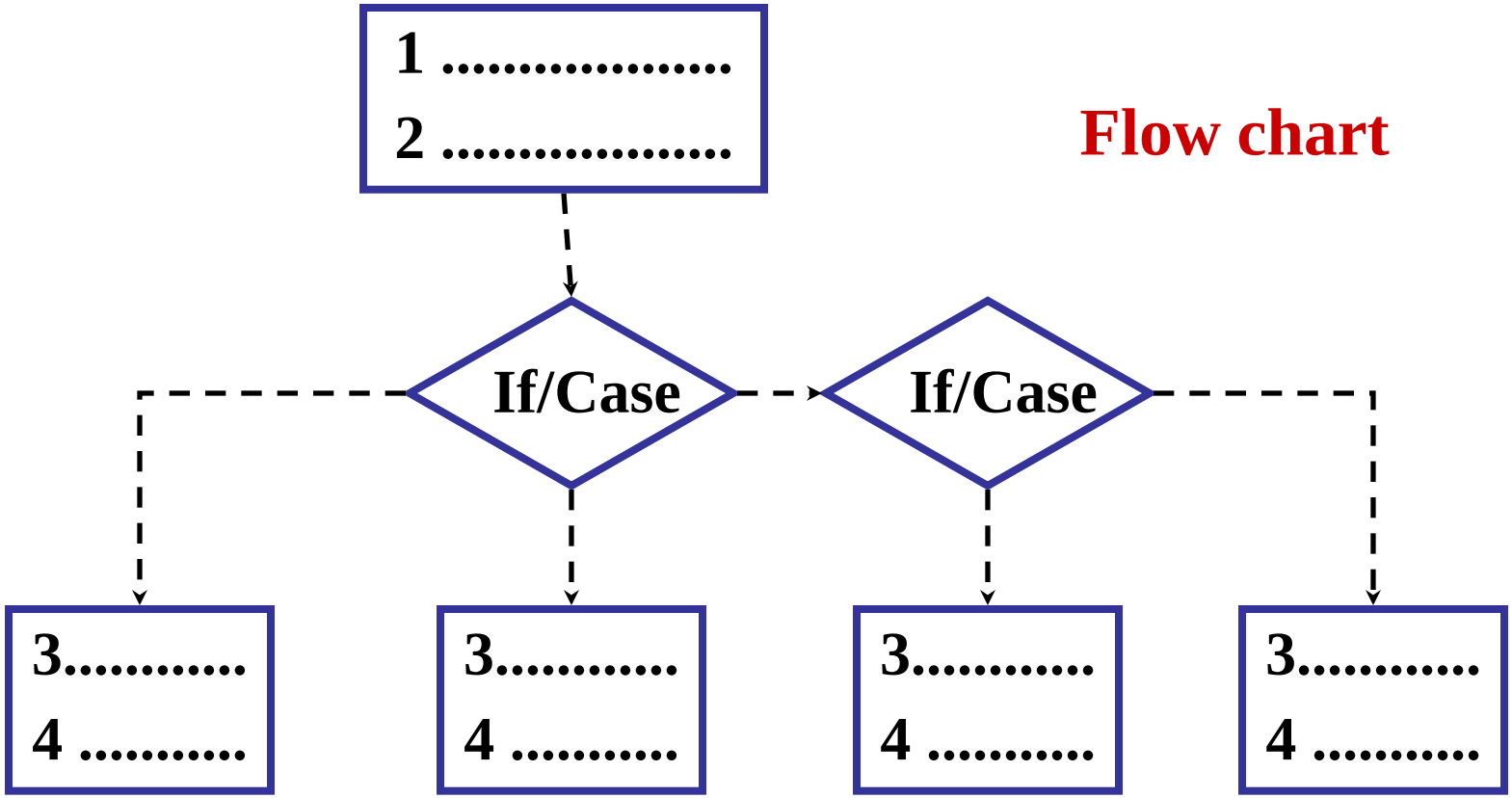
  ii) Translate the algorithm into a computer language (code) with a certain syntax, e.g. VBA in our case.

  iii) Test (debug) your program thoroughly.

- A good way of understanding what the structure of a program needs to be before actually writing the code is to draw a flow chart. A flow chart is a graphic representation of the program structure. You do not need to write all comments in detail, but it suffices to write general statements in words.

- It looks more or less like this…

Flow chart



The IF-structure in VBA

- The IF-structure allows you to change the flow of your program depending on various conditions. The logic of this structure is very similar to the discussed Excel built-in IF-function.

Syntax1: **If** condition **Then**

[statements]

[**ElseIf** condition **Then**

[elseifstatements]]...

} can be repeated many times

[**Else**

[elsestatements]]

End If

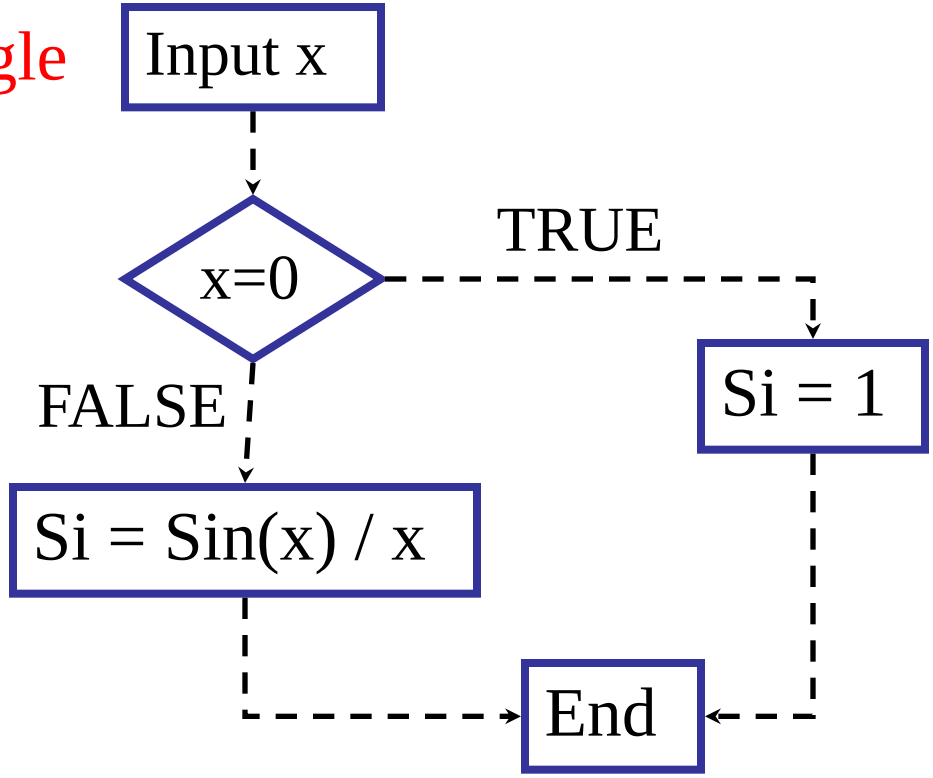
- condition( ) ≡ expressions which are true of false
- statements ≡ valid VBA commands
- elseifstatements ≡ executed when condition is true
- elsestatements ≡ executed when no previous condition is true

- Examples:

a) Write a UDF which produces the function

$$\mathrm{Si}(x) = \begin{cases} \frac{\sin x}{x} & \text{for } x \in \mathbb{R} \setminus 0 \\ 1 & \text{for } x = 0 \end{cases}$$

```
Function Si(x As Single) As Single
    If x = 0 Then
        Si = 1
    Else
        Si = Sin(x) / x
    End If
End Function
```



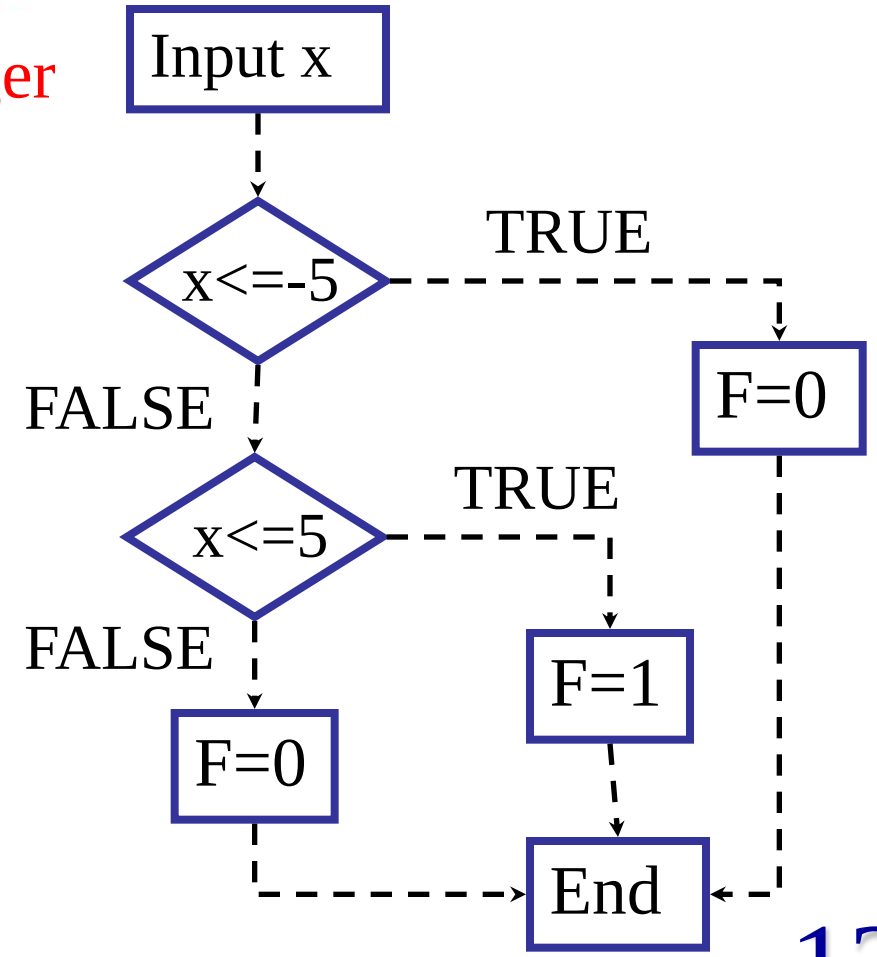
- Recall from Lab 2 Task 3 that this function also can be produced by using Excel built-in functions as

=IF(x=0,1,SIN(x)/x)

b) Write a UDF which produces the function

$$F(x) = \begin{cases} 0 & \text{for } x \le -5 \\ 1 & \text{for } -5 < x \le 5 \\ 0 & \text{for } x > 5 \end{cases}$$

```
Function F(x As Single) As Integer
    If x <= -5 Then
        F = 0
    ElseIf x <= 5 Then
        F = 1
    Else
        F = 0
    End If
End Function
```



- Recall lecture 3: =IF(A1>-5, IF(A1<=5,1,0) ,0)

c) Write a UDF which determines whether a certain date falls on a weekend or not!

```
Function WE(x As Date) As String
    Dim temp As Integer
    temp = Weekday(x)
    If temp = 1 Or temp = 7 Then
        WE = "That day falls on a weekend."
    Else
        WE = "That day is a weekday."
    End If
End Function
```

- Format the cell A1 as date and enter Now()
- "=WE(A1)" → That day is a weekday.
- Note that we declared all variable types.

Syntax2: **If** condition **Then** [statement1] : [statement2]: ...

- Just one line! The VBA statements are carried out when the condition is TRUE. Several statements are separated by ":"
- Expl.: The function F(x) can also be produced by:

```
Function Ftwo(x As Single) As Integer
    Ftwo = 1
    If x <= -5 Then Ftwo = 0
    If x > 5 Then Ftwo = 0
End Function
```

Syntax3: **IIf** (*condition*, value for true, value for false)

- Same syntax as for built-in functions with IF → IIF
- Expl.: The function in Expl. a) can also be produced by:

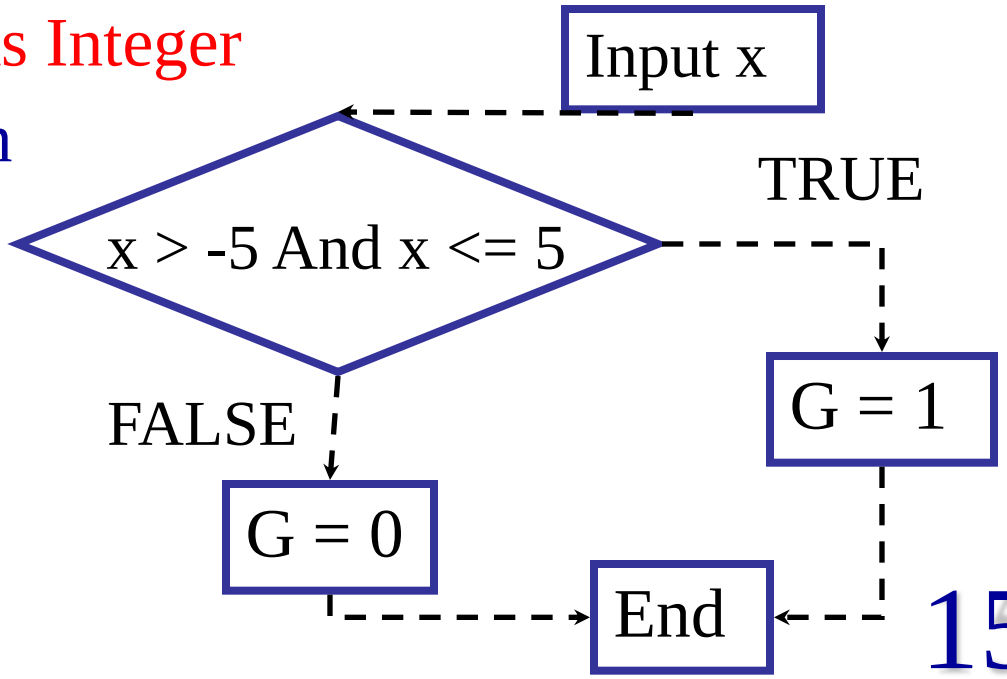
```
Function Fthree(x As Single) As Single
    Fthree = IIf(x = 0, 1, Sin(x) / x)
End Function
```

- Boolean operators
  - Just as for built-in functions one can use boolean operators to create more complex conditions.

    Syntax: condition1 **And** condition2 **And** condition3

    condition1 **Or** condition2 **Or** condition3
  - The logic is the same as for built-in functions.
  - Expl.: The function F(x) can also be produced by

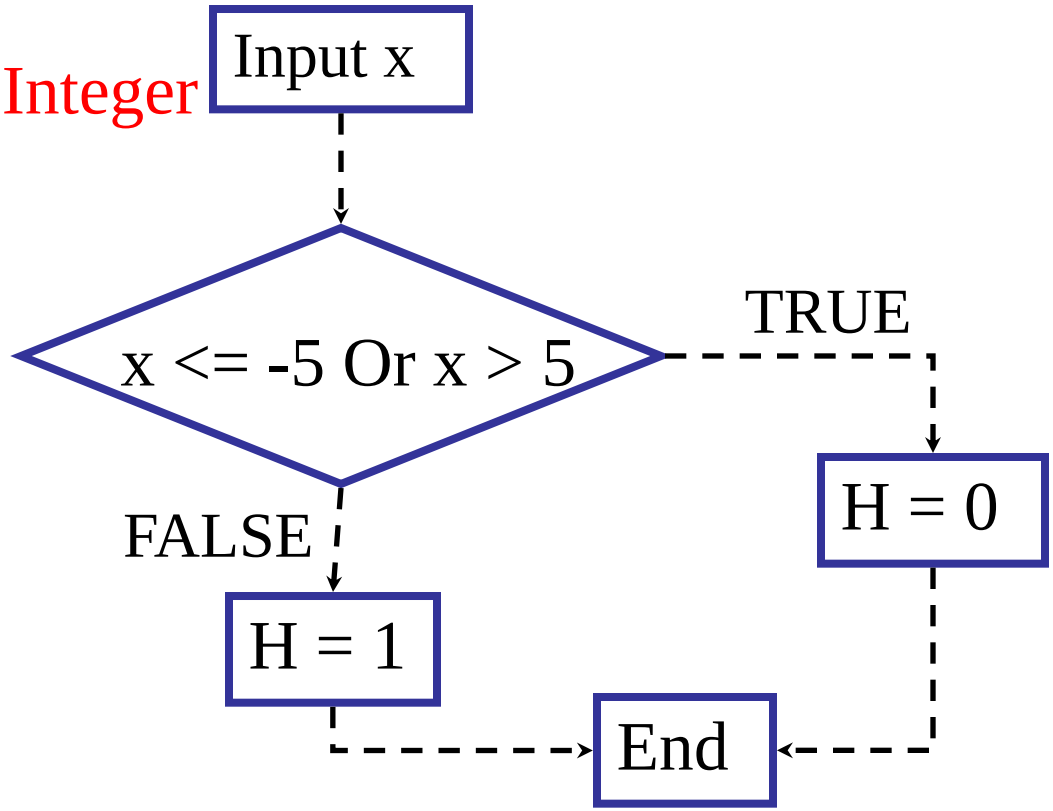
```
Function G(x As Single) As Integer
  If x > -5 And x <= 5 Then
      G = 1
  Else
      G = 0
  End If
End
```



  - We need to call it differently when it is on the same WS, e.g. G(x).

- Using “Or“ F(x) can be produced by

```
Function H(x As Single) As Integer
  If x <= -5 Or x > 5 Then
        H = 0
  Else
        H = 1
  End If
End Function
```



- You can also use “Not“ and produce the same logical structures as with built-in functions, e.g.

```
Function Fnot(x As Single) As Integer
      Fnot = IIf(Not (x <= -5 Or x > 5), 1, 0)
End Function
```

Example from resit exam 2007

$$f(x)=\begin{cases} 2, & \text{for } x \geq 1 \text{ Or } x \leq -1 \\ x+4, & \text{for } -1 < x < -0.5 \text{ Or } -0.5 < x < 0 \text{ Or } 0 < x < 1 \\ 0, & \text{for } x = 0 \text{ and } x = -0.5 \end{cases}$$

An If structure that produces this function is the following:

```
Function f(x As Single) As Single
If x >= 1 Or x <= -1 Then
        f = 2
Elseif x>-1 and x<-0.5 or x<0 and x>-0.5 or x<1 and x>0
        f = x+4
Else
        f = 0
  End If
End function
```

But this is not the only solution! There are in fact many ways of programming this function with VBA. For example:

```
Function f(x As Single) As Single
 If x >= 1 or x <= -1 Then
        f = 2
Elseif x=-0.5 or x=0 then
        f = 0
Else
       f = x+4
  End If
End function
```

This is a simpler function than the one in the previous page, but both are valid and equivalent.