Week 5
Control Structures for Programming: Selection, Functions
The Story So Far

You should have *some* understanding of

- Subprogram structure: `Sub Foo() ... End Sub`
- Variable declarations: `Dim myvar As Double`
- Types: `Double`, also `Long`, `Boolean`, `String`
- Assignment statements: `myVar = -1`
- Arithmetic expressions:
  - `myVar = 2*yourVar + herVar/3.5 - 1.6`
- Mod and `\` operators
- Running programs from the editor and menus
- Displaying results:
  - `MsgBox "Result is " & myVar`
Examples of control

If too hot
increase cold air flow
Otherwise too cold
decrease cold air flow

If frontal collision detected
trigger the air bag

Control is used extensively in computer programs:
- Vehicle stability control
- Precision manufacturing
- Chemical process control
- and more

Modern luxury vehicles contain many computer programs (about 100 million lines of code)

http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code
Fault tolerant control

- Example: How to keep the flight going when the plane has experienced severe damages?
- Automatic fault diagnosis and control

- Normal
- Damage 1
- Damage 2
- Damage n

Normal control → Algorithm 1 → Algorithm 2 → Algorithm n

Chowdhary et al. Autonomous Guidance and Control of Airplanes under Severe Damage, 2011
Program State

- The **program state** is the set of active variables and objects, external device state plus current execution position.
- Control structures let the programmer cause actions to occur (or recur) based on **decisions** about the program state.
- Main control structures are
  - **procedures** (for grouping subtasks)
  - **selection** (do *something* or perhaps do *something else*)
  - **iteration** (do *something* as long as required)
Boolean Expressions

• All decisions are represented by **Boolean expressions**
• Boolean expressions evaluate to True or False
• Involve **relational operators** to compare values

<table>
<thead>
<tr>
<th>Basic:</th>
<th>=</th>
<th>&lt;&gt;</th>
<th>&lt;</th>
<th>&lt;=</th>
<th>&gt;</th>
<th>&gt;=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maths:</td>
<td>=</td>
<td>≠</td>
<td>&lt;</td>
<td>≤</td>
<td>&gt;</td>
<td>≥</td>
</tr>
</tbody>
</table>

• Examples:

  2 < 3  
  num1 >= num2  
  intSum <> 0

  strName <= "Zanzibar"  
  ' string comparison

  Now() > "2014-08-25"  
  ' date comparison

  j = 3  
  strID = "A400"

• context determines whether the last two are relational (comparison) expressions or assignment statements. We will use the sub-program DemoBoolean() to understand this.
DemoBoolean()

- We will also use this program to illustrate how to use break point.

```vba
Sub DemoBoolean()
    Dim boo1 As Boolean
    Dim boo2 As Boolean
    Dim x As Integer

    'Initialise x as 5
    x = 5

    'You can assign True or False to a Boolean expression
    boo1 = True

    'Two Boolean expressions
    boo1 = (x > 7)

    ' boo2 = (x = 5)
    ' ^An assignment      ^ A comparison: Is x equal to 5?
    boo2 = (x = 5)
End Sub
```

Click in the margin to set a break point.
Selection If/Then/Else

- Symmetric form:

    **If** boolean-expression **Then**

    **statement-list1**

    **Else**

    **statement-list2**

    **End If**

- **boolean-expression** is evaluated
  - If it evaluates to True, **statement-list1** is executed
  - Otherwise, **statement-list2** is executed
The If/Then statement is used to make decisions using Boolean expressions.

Simplest form:

```
If boolean-expression Then
  statements
End If
```

- boolean-expression is evaluated
  - If it evaluates to True, statements are executed
  - otherwise (i.e., it must be False) skip over statements and continue with rest of program

Also referred to as a conditional statement.
Absolute value

- We will show two implementations of computing the absolute value, using
  - If/Then/Else
  - If/Then

\[ |x| = \begin{cases} 
  x & \text{if } x \geq 0 \\
  -x & \text{otherwise} 
\end{cases} \]
Absolute value: If/Then/Else example

```java
If intVal >= 0 Then
    intAbsVal = intVal
Else
    intAbsVal = -intVal  ' unary minus*
End If

' * much preferred to these change-sign equivalents:
    intVal = intVal * -1  ' yuk!
    intVal = 0 - intVal  ' double yuk!
```

*Code in Abs1*
If/Then example

```plaintext
intAbsVal = intVal
If intAbsVal < 0 Then
  intAbsVal = -intAbsVal  ' unary minus*
End If

' now we know that intAbsVal = |intVal|
' regardless of whether intVal is positive or negative

Code in Abs2
```
Examples

' Larger value
If a > b Then
  max = a
Else
  max = b
End If

's = sign of a (+1,-1,0)
If a <> 0 Then
  If a > 0 Then
    s = 1
  Else
    ' a < 0*
    s = -1
  End If
Else
  s = 0
End If

' (chained form left as exercise)
' * justified a few slides later

Code in MaxOf
More complex form

Chained form, symmetric form generalised to \( n \):

```
If boolean-expression1 Then
  statement-list1
ElseIf boolean-expression2 Then
  statement-list2
ElseIf boolean-expression3 Then
  statement-list3
  ...
Else
  statement-list-n
End If
```

Mantra: indent all statement lists embedded in a selection statement
Chained Selection example: Classification

Often need to classify a value based on ranges, such as deriving UNSW grade from mark (excluding PC):

' Precondition (assumption): 0 <= mark <= 100

If mark >= 85 Then
  grade = "HD"
ElseIf mark >= 75 Then  ' And Not (mark >= 85)
  grade = "DN"
ElseIf mark >= 65 Then
  grade = "CR"
ElseIf mark >= 50 Then
  grade = "PS"
Else  ' Not (mark >= 50), so mark < 50
  grade = "FL"
End If
Chained Selection example: Classification

Often need to classify a value based on ranges, such as deriving UNSW grade from mark (excluding PC):

```
If mark >= 85 Then
    grade = "HD"
ElseIf mark >= 75 Then
    grade = "DN"
ElseIf mark >= 65 Then
    grade = "CR"
ElseIf mark >= 50 Then
    grade = "PS"
Else
    grade = "FL"
End If
```

Explaining the principle:

A mark may satisfy multiple conditions. Only the statement associated with the **first** satisfied condition is executed.

Example:
A mark of 76 satisfies:
- mark >= 75
- mark >= 65
- ...

“mark >= 75” is the first satisfied clause, so ONLY execute grade = “DN”
Classification order

When chained selection is used to identify which range a value lies in, the best strategy is always to start from the highest (or lowest) threshold and work progressively to the other end

- \( n \) possible ranges have \( n-1 \) boundaries
- each boundary value is mentioned once only
- UNSWgrade is exactly like this (5 grades, 4 comparisons)

Each decision divides the space into two parts

The module GradeMU demonstrates what can go wrong. Beware: GradeMU contains errors.
Boolean Expressions

Boolean expressions comprise

- Boolean constants  \textit{True} \textit{False}
- Boolean variables
- Relational operations (comparisons giving Boolean values)
- Boolean operators
  - \texttt{binary}  \texttt{And} \texttt{Or} \texttt{Xor}
  - \texttt{unary}  \texttt{Not}
Truth Tables

Truth tables establish meaning of operators by enumerating each combination of operands and showing what the operation yields.

*Notation:* $T = \text{True}$, $F = \text{False}$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>A And B</th>
<th>A Or B</th>
<th>A Xor B</th>
<th>Not A</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

- **And** – both True
- **Or** – either True
- **Xor** – either (but not both) True
- **Not** – complement
Examples

\[ x \geq 0.0 \text{ And } x < 1.0 \]

' \( x \) lies between 0 (inclusive) and 1 (exclusive)

\[ a = b \text{ And } b = c \]

' all three are the same

\[ j < 0 \text{ Or } j > 0 \]

' equivalent to ????

\[ x \geq 0.0 \text{ Or } x < 1.0 \]

' True (why?)

\[ \neg (y \geq 0) \]

' equivalent to ????

\[ \neg \text{ found} \]

' complement of variable

\[ \text{found} = \text{True} \]

' redundant, just use found

\[ \neg (E1 \text{ And } E2) \]

' \((\neg E1) \text{ Or } (\neg E2)\)

\[ \neg (E1 \text{ Or } E2) \]

' \((\neg E1) \text{ And } (\neg E2)\)

' => de Morgan's Laws
De Morgan’s Law example

- $P = \text{Pressure}, \ T = \text{temperature}$

- $E_1: \ P \geq 300$
- $E_2: \ T \geq 200$

- Plant is safe if neither pressure exceeds 300 nor temperature exceeds 200
- Plant is safe if $\text{NOT} \ (E_1 \text{ or } E_2)$
De Morgan’s Law example

- E1: $P \geq 300$
- E2: $T \geq 200$
- Plant is safe if NOT (E1 or E2)

OR is the same as union
De Morgan’s Law example

- E1: \( P \geq 300 \)
- E2: \( T \geq 200 \)
- Plant is safe if NOT (E1 or E2)
- NOT (E1 or E2) = (NOT E1) AND (NOT E2)

AND is the same as intersection
Boolean Reasoning

Recall the \textit{sgn} $a$ example:

\begin{verbatim}
If $a \not= 0$ Then
  If $a > 0$ Then
    $s = 1$
  Else \quad ' a < 0 '*
    $s = -1$
...
\end{verbatim}

How did we know that $a < 0$ at the point shown?

- We've passed the first condition, so $a \not= 0$
- But we've failed the second condition, so \textbf{Not} $(a > 0)$

Hence we have $a \not= 0$ \textbf{And} \textbf{Not} $(a > 0)$

$= a \not= 0 \text{ And } a \leq 0$ \quad ' by definition of \textbf{Not}

$= a < 0$ \quad \quad ' by definition of \textbf{<>} and \textbf{<=}
Exercise: Leap year

- Leap year
  - A leap year must be divisible by 4
  - But not all centuries are leap years
  - Centuries are leap years only if they are divisible by 400
    - 2000, 2400 are leap years
    - 2100, 2200, 2300 are NOT leap year

- You want to write a program to classify whether a given year is a leap year

```plaintext
year = 2000
If Boolean-expression Then
   MsgBox “Yes, a leap year”
Else
   MsgBox “No, not a leap year”
End If
```

Can you come out with the Boolean expression?

You may need AND, OR, NOT.
The following test whether the variable `year` is divisible by 4:

\[(\text{year Mod 4} = 0)\] is True if year is divisible by 4, otherwise False.
Leap years

All years

(year mod 400 = 0)

year divisible by 4 but not 100
Leap years

All years

Mod 4 = 0

Centuries

2004

2100

2000

Mod 400 = 0

(year mod 400 = 0)

OR

Question: Say, you’ve written the program and want to test it. What years should you use to test the program?
**Functions**

- **Functions** are procedures (named groups of statements) that evaluate and return a **single value** of a specified type
  
  (**subprograms** perform actions but don't return anything)

- OO Basic functions can be used in formulas in sheet cells, or called from other parts of a program

- Declaration is always

  ```
  Function name(parameters...) As type
      declarations for local variables
      ...
      statements, including somewhere...

  name = expression
  End Function
  ```

(examples will be provided)
Parameters

- Values are passed to a function (or subprogram) via parameters
- Look like variable declarations (and are)
- Corresponding values in call: arguments

Code in Bigger

Module1

Function bigger(v1 As Double, v2 As Double) As Double
  If v1 >= v2 Then
    bigger = v1
  Else
    bigger = v2
  End If
End Function

1. formula uses Basic function
2. values copied to parameters v1 and v2
3. function begins execution
4. return value assigned to function name
5. result copied to cell containing formula
Built-in Functions

- OO Basic includes a range of mathematical, string and other functions
- Mathematical functions include
  - Sin, Cos, Tan, Atn (arctangent), all in radians
  - Abs, Sgn, Int (largest int ≤), Fix (truncate towards 0), Round (to any number of decimal places, including neg)
  - Sqr (√), Exp (e^x), Log (ln x)
  - Rnd (pseudo-random number)
- Other functions
  - IsNumeric(var) returns True if var is a number or can be converted to a number, otherwise False
  - Now() returns the current time as a date value
  - Format, allows numeric and other values to be formatted according to a pattern
Built-in Functions, continued

- Can also use functions used in worksheet formulas, with the notation
  \( \text{WorksheetFunction.name(args...)} \)
- Examples:
  \[
  \text{angle} = \text{WorksheetFunction.Radians(-90)}
  \]
  \[
  \text{angle} = \text{WorksheetFunction.atan2(x, y)}
  \]
  \[
  \text{payment} = \text{WorksheetFunction.PMT(rate, numPeriods, curValue)}
  \]

Requires **Option VBASupport 1** at the top of the module
Also works with cell values and ranges (next week)
UNSW Grade Function

- Embed algorithm (earlier slide) in function
- One parameter, mark (type is Integer), returns String
- Use on worksheet, fill down formula
- What if mark is out of range? Or not a number? (complete solution must cover all cases, Variant type enables this)
UNSWgrade with error checks

Function UNSWgrade(mark As Variant) As String

    If Not IsNumeric(mark) Then
        UNSWgrade = "Not a mark"
        Exit Function
    End If

    If mark < 0 Or mark > 100 Then
        UNSWgrade = "Mark out of range"
        Exit Function
    End If

algorithm coded as before, can rely on valid mark range;
change variable name grade to function name UNSWgrade

End Function

Note early return after detecting error condition and assigning result;
IsNumeric (Basic) vs IsNumber (worksheet). So much for consistency!
Quiz: Can you improve this program?

• Suppose someone wants to apply the Z-formula (a made-up formula):

\[ Z = H\left(\frac{1}{273^2} - \frac{1}{T^2}\right) \]

• to three different pairs of (H,T)
  – H = 10000; T = 249
  – H = 12000; T = 271
  – H = 14000; T = 295

• The person wrote the program on the next slide. What can be done better?
Option Explicit
Const T0 = 273

Sub Modular()

    Dim H1 As Double, H2 As Double, H3 As Double
    Dim T1 As Double, T2 As Double, T3 As Double
    Dim Z1 As Double, Z2 As Double, Z3 As Double

    H1 = 1e4: H2 = 12e3: H3 = 14e3:
    T1 = 249: T2 = 271: T3 = 295:

    Z1 = H1*(1/T0^2 - 1/T1^2)
    Z2 = H2*(1/T0^2 - 1/T2^2)
    Z3 = H3*(1/T0^2 - 1/T3^2)

    MsgBox "Z1 = " & Format(Z1,"0.000") & "; " & _
        "Z2 = " & Format(Z2,"0.000") & "; " & _
        "Z3 = " & Format(Z3,"0.000") & "."

End Sub
Modular structure

- Identical work should be performed by a function rather than repeating the code
- Makes code easier to maintain and understand
  - Hide the details of the calculation
- Let us use a function to calculate the Z-formula this time

Code in Modular
Quiz: Do these two functions do the same thing?

Function Func1(x As Double) As Double

    Func1 = 3 * x + 1

End Function

Function Func2(x As Double) As Double

    x = 3 * x
    Func2 = x + 1

End Function
Let us see what the differences are

Sub RunFunc1()
    Dim x As Double
    Dim y As Double

    x = 2
    y = Func1(x)

    MsgBox "y = " & y
    MsgBox "x = " & x
End Sub

Function Func1(x As Double) As Double
    Func1 = 3 * x + 1
End Function

Sub RunFunc2()
    Dim x As Double
    Dim y As Double

    x = 2
    y = Func2(x)

    MsgBox "y = " & y
    MsgBox "x = " & x
End Sub

Function Func2(x As Double) As Double
    x = 3*x
    Func2 = x + 1
End Function
What if I change the identifier from \( x \) to something else in the function?

Sub RunFunc2()

Dim x As Double
Dim y As Double

x = 2
y = Func2(x)

MsgBox "y = " & y  
  y = 7
MsgBox "x = " & x  
  x = 6

End Sub

Function Func2(x As Double) As Double
    x = 3*x
    Func2 = x + 1
End Function

Sub RunFunc3()

Dim x As Double
Dim y As Double

x = 2
y = Func3(x)

MsgBox "y = " & y  
  y = 7
MsgBox "x = " & x  
  x = 6

End Sub

Function Func3(notX As Double) as Double
    notX = 3*notX
    Func3 = notX + 1
End Function
Sub RunFuncByVal()

    Dim x As Double
    Dim y As Double

    x = 2
    y = FuncByVal(x)

    MsgBox "y = " & y
    MsgBox "x = " & x

End Sub

Function FuncByVal(ByVal x As Double) as double

    x = 3*x
    FuncByVal = x + 1

End Function

Sub RunFuncByRef()

    Dim x As Double
    Dim y As Double

    x = 2
    y = FuncByRef(x)

    MsgBox "y = " & y
    MsgBox "x = " & x

End Sub

Function FuncByRef(ByRef x As Double) As Double

    x = 3*x
    FuncByRef = x + 1

End Function
### ByVal

**Sub** RunFuncByVal()

```vba
Dim x As Double
Dim y As Double

x = 2
y = FuncByVal(x)
MsgBox "y = " & y
MsgBox "x = " & x
```

**End Sub**

**Function** FuncByVal(ByVal x as double) as double

```vba
    x = 3*x
    FuncByVal = x + 1
End Function
```

To understand ByVal, we need to see what happens inside computer memory.

<table>
<thead>
<tr>
<th>Memory space for RunFuncByVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
</tr>
<tr>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory space for FuncByVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
</tbody>
</table>

If ByVal is used, the function allocates a separate memory space for the variable. The “x” in the sub is different from the “x” in the function and is not affected by the “x” in function.
### ByRef

If `ByRef` is used, the function is given permission to manipulate the variable specified.

**Memory space for RunFuncByRef**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>7</td>
</tr>
<tr>
<td>x</td>
<td>2</td>
</tr>
</tbody>
</table>

**Memory space for FuncByRef**

No memory is allocated for \( x \) and the function is allowed to manipulate \( x \) in the sub.

```vbc
Sub RunFuncByRef()
    Dim x As Double
    Dim y As Double

    x = 2
    y = FuncByRef(x)

    MsgBox "y = " & y
    MsgBox "x = " & x
End Sub

Function FuncByRef(ByRef x As Double) As Double
    x = 3 * x
    FuncByRef = x + 1
End Function
```
Quiz: What are the values of x and y at the end of the sub?

Sub RunFuncByRefByVal()

    Dim x As Double: Dim y As Double: Dim z As Double

    x = 2: y = 10
    z = FuncByRefByVal(x,y)

    MsgBox "z = " & z
    MsgBox "x = " & x
    MsgBox "y = " & y

End Sub

Function FuncByRefByVal(ByRef x As Double, ByVal y As Double) As Double
    x = 3*x
    y = 5*y
    FuncByRefByVal = x + y
End Function
Parameters revisited

• Values passed to procedures are called **arguments**
• Variables defined to hold these values are called **parameters**
• Parameters are local variables, like ones declared using `Dim`

• Data can be passed in two different ways:
  – **by value** (argument is expression, only value transferred)
  – **by reference** (argument is variable, parameter acts as a temporary alias)
  – keywords `ByVal` and `ByRef` specify which

• If a function is called from a worksheet, **ByVal** is always use.
• If a function/sub is called from another function/sub, **ByRef** is assumed if omitted
Example: Swapping values

Sub Swap(ByRef v1 As Variant, ByRef v2 As Variant)
    Dim tmpVal As Variant
    tmpVal = v1
    v1 = v2
    v2 = tmpVal
End Sub

y = -23.5: z = 6
Call Swap(y, z) ' what are the values of y and z?

We used the **Variant** type so the procedure can be used to exchange the contents of variables any fundamental type, except objects. This is the only recommended use for Variants.

**Motivation:** imagine exchanging the positions of an apple and an orange on the table in front of you.

Now do it again using just one hand!
Example: Parachutist Velocity Profile

Velocity of an object of mass $m$ falling under the influence of gravity and subject to drag is

$$v(t) = v_0 e^{-(c_d/m)t} + \frac{g m}{c_d} (1 - e^{-(c_d/m)t})$$

Where $v_0$ is the initial velocity, $g$ is acceleration due to gravity (m s$^{-2}$) and $c_d$ is the drag coefficient (in kg s$^{-1}$).

The exponential factor decays in magnitude, so the velocity asymptotically approaches $g m / c_d$.

For a free-falling 70kg parachutist with $c_d = 12.5$, this **terminal velocity** is $\sim 55$ m s$^{-2}$ (200km/hr).
Parachutist Velocity Profile, continued

When the parachutist pulls the rip-cord \((t = t_c)\), the same equation applies subsequently except that...

- \(v_0\) is the velocity at the instant the parachute opens
- there is a new, larger, drag coefficient \(c_{dp}\)
- the time parameter is \(t - t_c\) instead of \(t\)
Parachutist Simulation

- We can write a function that, given all parameters, calculates the velocity at any time \( t \)
- The function can be filled down a worksheet
- The algorithm, expressed in *pseudocode*, is

```plaintext
If \( t < t_c \) Then  ' still in free-fall
  Calculate free-fall velocity
Else
  Calculate velocity at time \( t_c \)
  Store in velocityChute
  Calculate velocity using cdp and velocityChute
End If
```
How can you improve this implementation?

Function VelX(t, g, m, cd, v0, tc, cdp)
    If t <= tc Then
        'Compute velocity of freefall
        VelX = v0 * Exp(-cd / m * t) _
            + g * m / cd * (1 - Exp(-cd / m * t))
    Else
        ' compute velocity when parachute is first deployed
        v0p = v0 * Exp(-cd / m * tc) _
            + g * m / cd * (1 - Exp(-cd / m * tc))
        ' compute velocity after parachute is deployed
        VelX = v0p * Exp(-cdp / m * (t - tc)) _
            + g * m / cdp * (1 - Exp(-cdp / m * (t - tc)))
    End If
End Function
1. The decay expression $e^{-(c_d/m)t}$ has its own function:

```
Function Decay(drag As Double, mass As Double, _
    time As Double) As Double

    Decay = Exp(-drag * time / mass)

End Function
```

2. The terminal velocity has a named variable (to make the expression more meaningful). We’ve called it `velocityLimit`.

3. The structure of the algorithm (before/after chute, common components) makes the solution easy to follow, especially compared to the massive IF formula that would be required without the coded function.
Another Worked Example

- Fluid flow: calculating the Moody friction factor in a pipe with a circular cross-section and certain properties, carrying fluids of various kinds
- Solution space has three distinct regions, requiring a classification of the flow type
- We use the model described at http://www.lmnoeng.com/moody.php
- Other references on sheet
Advanced language features that are not essential knowledge (but useful to know) are marked A
Multiway selection with a common expression: VBA has a simpler way than many *If* statements

**Select Case**

Select Case *expression*

1. **Case** *value*
   - statements
2. **Case** *value₁*, *value₂*
   - statements
3. **Case** *value₁ To value₂*
   - statements
4. **Case** *Is > value*
   - statements
5. **Case** *Like pattern*
   - statements
6. **Case** *Else*
   - statements
7. **End Select**

*expression* is evaluated first, then matched against each label.

- Single value
- Multiple values grouped
- Inclusive range of values
- Open range of values
- Values matching pattern
- All unmatched values

Any number of each of these can occur.
Select Case example

• Classify a single-character string c

Select Case UCase(c)
  Case "A","E","I","O","U"
    sym = "vowel"
  Case "0" To "9"
    sym = "digit"
  Case "A" To "Z" ' or Case Like "[A-Z]"
    sym = "consonant"
  Case ""
    sym = "empty"
  Case Like "??*"
    sym = "more than one char"
  Case Else
    sym = "punctuation"
End Select

use of UCase saves repeating upper and lower case letter rules
Strings

• A **string** is a sequence of characters
• Strings in OO Basic can be of virtually any length (including zero, of course)
• Components are **characters**
  – represented internally as codes with numeric value from 0 to 255*
  – each code is interpreted by an output device as a graphic character or special action
  – functions **Chr** and **Asc** convert between character and numeric code

* For conventional ASCII characters, OO Basic also supports Unicode characters.
Literal Strings

• String constants are enclosed in double-quote characters "..." (not single quotes)

• How to include a double-quote as a member of the string (the delimiter-as-data problem)? Write it twice, as in

"Reference: ""OpenOffice for Experts""

• Empty string is denoted by ""

• & concatenates (joins) two strings
  - "c" & "at" (= "cat")    "cat" & "ch" & "-22"
String Comparison

- relational (comparison) operators
  - compare individual characters, so
    "cat" < "dog"  (because "c" < "d")
    "cat" < "cave"  (because "t" < "v")
    "cat" < "catastrophe"  (prefix)
    "Cat" < "cat"  (case sensitive)
  - can change to case-insensitive comparison
    (to make "Cat" = "cat") by placing
      Option Compare Text
    at the top of the module
String Matching

- The **Like** operator allows a string expression to be tested against a pattern that includes wildcards

  \[
  \text{strData Like "pattern"}
  \]

- The pattern consists of
  
  - `?` any single character
  - `#` any digit character
  - `*` zero or more characters
  - `[list]` any character in \(list\)
  - `[!list]` any character not in \(list\)
### String Matching Examples

Assume the following values

\[ \text{str1} = \text{"cat"}; \ \text{str2} = \text{"Route66"}; \ \text{str3} = \text{"2*3"} \]

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>str1 Like &quot;cat&quot;</td>
<td>True</td>
<td>ordinary chars OK</td>
</tr>
<tr>
<td>str1 Like &quot;c?t&quot;</td>
<td>True</td>
<td>? matches the character a</td>
</tr>
<tr>
<td>str1 Like &quot;c&quot;</td>
<td>False</td>
<td>must match whole of string</td>
</tr>
<tr>
<td>str2 Like &quot;*##&quot;</td>
<td>True</td>
<td>ends in two digits</td>
</tr>
<tr>
<td>str2 Like &quot;<em>[A-Z]</em>&quot;</td>
<td>True</td>
<td>([A-Z]) matches any capital</td>
</tr>
<tr>
<td>str2 Like &quot;*![6]&quot;</td>
<td>False</td>
<td>can't match non-6 at end</td>
</tr>
<tr>
<td>str2 Like &quot;<em>![6]</em>&quot;</td>
<td>True</td>
<td>(you work it out!)</td>
</tr>
<tr>
<td>str3 Like &quot;*3&quot;</td>
<td>True</td>
<td>* matches 2*</td>
</tr>
<tr>
<td>str3 Like &quot;#[*]?&quot;</td>
<td>True</td>
<td>to match special chars * [ \ ? literally, enclose in [ ]</td>
</tr>
</tbody>
</table>
Summary

- Boolean expressions represent the result of decisions
- If-Then used for selection (three forms)
- Functions return a single value, can accept any number of parameters
- Parameters are local variables that are given initial values when the function is called
- OO Basic functions can be used in worksheet formulas
- Interpreter can be used to trace a program to assist in debugging