Iteration (Repetition)

- Very often need to execute statements repeatedly
- Loops are statements that can do this
- Process is called iteration
- Kinds of loop:
  - For (iterate a fixed number of times)
  - While (iterate as long as something is True)
  - Do-Loop (general iteration)

For loop examples

```
1. factorial (assume n is set beforehand)
   fact = 1
   For i = 1 To n
     fact = fact * i
   Next i

2. sum = sum of the first n squares (reverse order!)
   sum = 0
   For k = n To 1 Step -1
     sum = sum + k^2
   Next k

3. Compound interest without exponentiation
   (principal P, interest rate r (%) for n years)
   balance = P
   For y = 1 To n
     balance = balance * (1.0 + r*0.01)
   Next y
```

Iteration – For

- For statement sets an integer loop variable to each value in a range and executes statements forming the loop body for each value:

```
For intvar = start To finish [Step amount]
  statements
Next intvar
```

- start and finish are expressions, evaluated once
- Step amount is optional (default 1)
- statements may use (but not change) the value of the loop variable
- loop must terminate (VBA checks consistency of start, finish and amount)

Iteration – While

- While statement continues to execute statements as long as a Boolean expression is True

```
While boolean-expression
  statements
Loop body
Wend
```

- Loop guard is evaluated
  - If it is True execute the loop body and go back to start of loop to re-test the guard
  - Otherwise (i.e., it is False) exit loop and continue with rest of program
- Loop body must change state so that loop guard can eventually become False (else infinite loop

Why not End While? Beats me.
Iteration – termination

• Loop body must change state so that loop guard can eventually become False (else we have produced an infinite loop)

While loop examples

1. largest power of 2 less than some limit (lim)
   pow2 = 1
   while 2*pow2 <= lim
     pow2 = 2*pow2
   wend

2. approximation to square root: √k for any k ≥ 0
   Alg: Newton-Raphson. For theory see Wikipedia or
   http://www.sosmath.com/calculus/diff/der07/der07.html
   \[ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \]
   where \( f(x) = x^2 - k \) and thus \( f'(x) = 2x \)
   Dim dblx As Double ' current estimate
   Dim dblLastx As Double ' previous estimate
   dblx = k: dblLastx = 0 ' guesses
   While Abs(dblx – dblLastx) > EPSILON
     dblLastx = dblx
     dblx = dblx - (dblx ^ 2 – k) / (2 * dblx)
   wend
   dblx ≈ √k

Pseudocode and refinement

How did we come up with the square root solution?
• Initial version is statement of purpose
  set dbl to within epsilon of \( \sqrt{k} \) using Newton-Raphson algorithm
• Express algorithm using English statements and VB control structures (if, while etc): pseudocode
• Indent subsidiary statements
• Refine to subprograms and VB statements

set initial guess and last guess to suitable starting values
while current guess and last guess are too far apart
  recalculate guess using N-R formula
wend

Pseudocode and refinement

Refined pseudocode: remember to update the variable holding the last guess before updating the current guess;
Also choose a safe value for the initial guesses

set current guess (dblx) to \( k \) [or \( 1, k/100, \text{any } > \text{EPSILON} \)]
set last guess (dblLastx) to 0
while the abs val of the difference is > EPSILON
  update dblLastx
  update dblx using the formula
wend

(demo: use this loop in a subprogram to show convergence)

Other kinds of loop

Do-Loop statement is similar to While, but the loop body is always executed at least once:

Do statements Loop body Loop guard
Loop While boolean-expression
Alternative form uses complemented guard:

Do statements Loop Until boolean-expression
Can terminate the loop early with the statement Exit Do (also Exit While and Exit For)

Excel object model (simplified)

• VBA programs can interact with the data on a worksheet, create new sheets etc.
• Workbook, sheets, cells form a hierarchy of objects, manipulated like ordinary variables (can use and/or change)
• Simplest reference to cells is via top-level Application object:
  Application.ActiveCell
  Application.ActiveSheet.Cells(row,col)
  - any integer expression for \( \text{row and col} \)
  - row counts from 1 to 65535*
  - col counts from 1 (= A) to 255*
• Can omit Application, Prefix

* Excel 2007+ accepts a wider range than this
## Excel object model, cont.

- Each cell has a large number of attributes or properties, recording its contents and appearance. Most can be changed.
- Most important properties from our viewpoint are Value and Formula, the two attributes easily changed by the user
  - Value is a variant (untyped value)
  - Formula is a string (as it would appear in the formula bar)
  - both can be assigned

## Worksheet Programming

- We can use values on the active worksheet as inputs to a program, and overwrite cells to show output and calculated values
- We can iterate over rows or columns of data
- We can detect empty cells to stop the iteration
- Example: running sum to the right of a column (defective subprocess over, to be corrected in the lecture)

```
' WARNING: contains a deliberate error
Sub AddRunningSum()
    Dim row As Integer ' index always integral
    Dim col As Integer
    Dim sum As Double ' numeric values normally real
    row = 2: col = 1: sum = 0
    ' note: separates multiple simple statements
    While ActiveSheet.Cells(row, col).Value <> ""
        sum = sum + ActiveSheet.Cells(row, col).Value
        ActiveSheet.Cells(row, col).Value = sum
    Wend
End Sub
```

## Example: Running sum

## Debugging Loops

- If a program fails to terminate, press Ctrl-Alt-Break (Break may be labelled Pause), or Esc in the labs
- VBA interpreter pauses, shows program state:

```
current execution point
```

## Procedures

- Functions and Subprograms are together called procedures
- Functions return a single value, should not change program state
- Subprograms can change state by
  - altering variables outside the procedure
  - communicating with other applications
  - changing the values of arguments passed by reference (next week)
- Function argument list enclosed in parentheses, subprogram arg list isn’t unless you use the call statement (more VB inconsistency)

## Sequential Algorithms

VBA is particularly useful for tasks such as
- Processing rows or columns on one pass
- Locating cells with particular characteristics
- Identifying extreme values (max/min)

These are called **sequential tasks**, because they treat the data as a sequence of items, usually stored in cells

Standard sequential algorithms can be adapted to solve these kinds of problems
Simple Scans

The simplest cases just process each element of a sequence in order (for convenience)

- Elements are independent
- End of sequence normally marked by empty cell

Pseudocode: column scan

```vba
set col to the required column number
set row to the row where the sequence starts
while ActiveSheet.Cells(row, col) is not empty
    apply required step to this cell
    move row one cell down
wend
```

State-based Scans

Other cases accumulate history in variables during the scan

- Element order may or may not matter
- Variables need to be correctly initialised
- Examples: `AddRunningSum`, maximum or minimum calculation, run of equal values, merging ordered sequences, mode of an ordered sequence

Maximum/minimum

- Max/min calculations maintain a "best so far" variable
- May also need to know the position of the (first or last occurrence of) the maximum

Pseudocode for maximum of a column:

```vba
set col to the required column number
set row to the row where the sequence starts
set max to the value in that cell
while ActiveSheet.Cells(row, col) is not empty
    if cell value is greater than max then
        set max to cell value
    end if
    move row one cell down
wend
```

Maximum/minimum, continued

If the problem is to highlight all extreme (max or min) values, need two passes

- First pass finds the extreme value (prev slide)
- Second pass processes each cell whose value matches the extreme value
- Next week we will see how cell appearance can be altered (more general than conditional formatting) – Macro Recorder used to capture relevant code

Example: local context

Task: Process a row of cells containing numeric values, identifying all peaks in the data (a peak is a value that is greater than those on both sides). Mark the peaks by writing a "^" in the cell below the peak. The first non-numeric cell ends the list.

Sample data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>-94</td>
<td>63</td>
<td>15.3</td>
<td>0</td>
<td>74.2</td>
<td>193</td>
<td>-64.2</td>
<td></td>
</tr>
</tbody>
</table>

(Solution developed in lecture)
Strings

- A **string** is a sequence of characters
- Strings in VBA 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

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: ""Excel for Experts""
- Empty string is denoted by ""

String Operators

- `&` concatenates (joins) two strings
  "c" & "at" (= "cat")  "cat" & "ch" & "-22"
- 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
  `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
- Same notation as Access' query partial match

String Matching Examples

Assume the following values

```
str1 = "cat": str2 = "Route66": str3 = "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?&quot;</td>
<td>True</td>
<td>matches the character a</td>
</tr>
<tr>
<td>str1 Like &quot;k&quot;</td>
<td>False</td>
<td>must match whole of string</td>
</tr>
<tr>
<td>str2 Like &quot;88&quot;</td>
<td>True</td>
<td>ends in two digits</td>
</tr>
<tr>
<td>str2 Like &quot;[A-Z]&quot;&quot;</td>
<td>True</td>
<td>[A-Z] matches any capital</td>
</tr>
<tr>
<td>str2 Like &quot;[16]&quot;&quot;</td>
<td>False</td>
<td>can't match non 6 at end</td>
</tr>
<tr>
<td>str2 Like &quot;[16]&quot;&quot;</td>
<td>True</td>
<td>(you work it out)</td>
</tr>
<tr>
<td>str3 Like &quot;3&quot;&quot;</td>
<td>True</td>
<td>matches 2*</td>
</tr>
<tr>
<td>str3 Like &quot;[1]&quot;&quot;</td>
<td>True</td>
<td>matches special chars *</td>
</tr>
</tbody>
</table>

Select Case

Multiway selection with a common expression: VBA has a simpler way than many `If` statements

```
Select Case Expression
    Case value
        statements
    Case value, value, value, statements
    Case value, To value, statements
    Case Is > value statements
    Case Like pattern statements
    Case Else statements
End Select
```
Select Case example

- Classify a single-character string \( c \)

```vba
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.

Built-in Functions

- VBA includes a range of mathematical, string and conversion functions.
- Accept one argument (sometimes other than one) and return exactly one value.
- Mathematical functions include:
  - \( \sin, \cos, \tan \), \( \tan^{-1} \) (arctangent), all in radians
  - Abs, Sgn, Int (largest int \( \leq \) value), Fix (truncate towards 0)
  - Sqr (\( \sqrt{\cdot} \)), Exp (\( e^x \)), Log (\( \ln \))
- Rnd (pseudo-random number)
- Look up Math Functions in VBE's Help.

Built-in Functions, cont.

- Can also use any function available on a worksheet, using `Application.WorksheetFunction.[name](args...)`
- Examples (Application can be omitted):
  - `dblAngle = WorksheetFunction.Radians(-90)`
  - `payment = WorksheetFunction.PMT(rate, numPeriods, _curValue)`
  - `code = WorksheetFunction.Replace(text, start, 3, _"***")`
- Some expect Range objects (described in a later lecture) as their data source.
- Look up Worksheet Functions in VBE's Help.

String and Character Functions

- Predefined names:
  - Scientific number in scientific notation
  - Percent integer as percentage
  - Currency number in $9,999.99 or [\$99.99] format
  - Long Date date in format defined in Control Panel (also Medium Date, Short Date)
  - Long Time time in format defined in Control Panel (also Medium Time, Short Time)
- Specialised formatting of numbers and dates:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Len(str)</td>
<td>number of characters in str</td>
</tr>
<tr>
<td>Mid(str,start,len)</td>
<td>len-char substring of str, pos @ start</td>
</tr>
<tr>
<td>Asc(str)</td>
<td>char code for first char in str</td>
</tr>
<tr>
<td>Chr(code)</td>
<td>Unit string containing char with code</td>
</tr>
<tr>
<td>LCase(str)</td>
<td>str with all upper-case chars converted to lower case (also UCase)</td>
</tr>
<tr>
<td>Trim(str)</td>
<td>str with spaces removed from both ends (also Trim, RTrim)</td>
</tr>
<tr>
<td>InStr(str,match)</td>
<td>first position in str where match occurs, or 0</td>
</tr>
<tr>
<td>Replace(str,match,repl)</td>
<td>str with all occurrences of match replaced by repl (other args possible)</td>
</tr>
<tr>
<td>String(str,n)</td>
<td>n copies of first char of str</td>
</tr>
</tbody>
</table>

Type Conversion Functions

- Can mix up numeric types (`dblAngle + intOffset` is evaluated as Double).

<table>
<thead>
<tr>
<th>Function name</th>
<th>Accepts as argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CStr</td>
<td>number</td>
<td>String (also <code>Format</code> next slide)</td>
</tr>
<tr>
<td>CInt</td>
<td>number or string</td>
<td>Long Integer (rounded if from Double)</td>
</tr>
<tr>
<td>CDbl</td>
<td>number or string</td>
<td>Double</td>
</tr>
<tr>
<td>VInt</td>
<td>string</td>
<td>Integer or Double, ignores trailing ( \pm )</td>
</tr>
<tr>
<td>Round(r,d)</td>
<td>number</td>
<td>( r ) Rounded to ( d ) decimal places</td>
</tr>
<tr>
<td>Hex</td>
<td>Integer</td>
<td>Hexadecimal string form</td>
</tr>
</tbody>
</table>

Format Function

- Specialised formatting of numbers and dates:
  - `Format(expr, fmt)` returns a formatted string.
  - Predefined formats can be used for \( fmt \):

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scientific</td>
<td>number in scientific notation</td>
</tr>
<tr>
<td>Percent</td>
<td>integer as percentage</td>
</tr>
<tr>
<td>Currency</td>
<td>number in $9,999.99 or $[99.99] format</td>
</tr>
<tr>
<td>Long Date</td>
<td>date in format defined in Control Panel (also Medium Date, Short Date)</td>
</tr>
<tr>
<td>Long Time</td>
<td>time in format defined in Control Panel (also Medium Time, Short Time)</td>
</tr>
</tbody>
</table>
User-defined formats assemble a string using codes for each part of the data

<table>
<thead>
<tr>
<th>Predefined code</th>
<th>Displays...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 9, 5</td>
<td>digit or zero</td>
</tr>
<tr>
<td>#</td>
<td>digit or nothing</td>
</tr>
<tr>
<td>%</td>
<td>percentage placeholder (×100, show %)</td>
</tr>
<tr>
<td>e+, e- E+, E-</td>
<td>scientific notation</td>
</tr>
<tr>
<td>@</td>
<td>single char or space</td>
</tr>
<tr>
<td>c</td>
<td>force lower-case or upper-case</td>
</tr>
<tr>
<td>ddd mmm yy hh nn</td>
<td>date/time components (see text)</td>
</tr>
</tbody>
</table>

See VBE Help for details

### Critical Path, continued

- To identify the CP and project length, need to find the longest task in each stage and add up their durations
- The difference between a task duration and the maximum for a stage is called **float time**
- Tasks on the CP have float time = 0
- So, intermediate aim is to calculate float times

### Assignment 1 (geo)

- Purpose: implement geometric algorithms on plane surfaces
- Data format: sheet with columns of eastings and northings (or x and y values in general)
- Task: check closure, count points, calculate perimeter, area, also centroid if time
- Required processes
  - Read from columns, detect end of data
  - Read/write access to cells by address
  - Gauss-Green formula (uses add/sub/mul/div only)
- What you submit must be your own work
- Support and tools
  - Map, with current point highlighted
  - Google-map-based web data point harvester
  - Dryrun checker, use prior to submission, one basic test only
  - Forum (general questions), consultants (specific one-one help)

### Summary

- Use pseudocode to express algorithm prior to coding in VB
- Active sheet and its cells can be addressed directly
- Sequential algorithms process a row or column of data systematically
- Some algorithms accumulate values, others process cells locally and order-independently
- Strings store simple character sequences
- Built-in functions evaluate commonly-required quantities (string and numeric)
- Relatively complex tasks can usually be subdivided into easily managed subtasks and solutions combined