ENGG1811 Computing for Engineers

Week 4
Introduction to Programming and OpenOffice.org Basic

Weeks 4-6 Programming with OpenOffice Basic
Notices – Week 4

• Mid-Session Exam
  - occupies the first part of the week 5 lab
  - can use OpenOffice Calc only, *no web access*
  - Documentation provided
  - answers in spreadsheet, submitted

• OO Basic Labs (weeks 5 to 7)
  - you must be prepared for each one
  - programming is not easy for many people
  - tutors will help with detail, but cannot teach you how to program
  - multiple tasks: subset OK (usually the first exercise) for students having difficulties
  - online assessment will require minimal effort
Resources

• The lecture notes should be sufficient
• There are references on OO Basic on the web. Go to the course website and click on Resources, and choose OO Basic references.
Automating spreadsheet computation

• In Week 1, we used spreadsheet to count the number of heart beats
  – You entered formulas in a few cells
  – You filled the formulas in other cells by dragging the mouse

• There are limitations to that
  – What if there are 1,000,000 rows of data?

• Programming allows you to automate the computation

• Demo file: week04demo-heartbeat.ods
Why learn programming?

• Many many good reasons
  – Automate computation
  – Learn a new way of thinking
    • Using computing and computation to solve problems
    • Computational thinking
  – Add a new language or an important skill to your repertoire
  – ...
  – Applying computing to areas that people have not thought about before
    • Plenty of potential to change the world
A tower of 1500 foam bricks
What is so special about this?
Assembled by 4 quadrocopters automatically
(Another application of computing and programming!)
Holy grail: Automatic building assembly

• The previous 2 pages show pictures of automatic assembly by flying robots
  – Video: http://vimeo.com/33713231

• Demonstration of an ultimate dream: Automatic building assembly
  – Pre-fabricate modules of the building (slabs, windows, doors etc.)
  – The robots assemble the modules following a plan (put module A in, then module B, …)

From plans to algorithms

• Plans, recipes, procedures, assembly instructions, ...
  – These are words/phrases that we use in our daily language to describe a set of instructions

• Algorithm: a set of instructions
  – But not any instructions, these are instructions meant for the computers or computation

• Algorithms are implemented in computer programs
  – Instructions are written in programming languages
  – Calc’s formulas are a limited kind of programming notation
Your Turn (#1, easy)

• Devise an algorithm to calculate the average of two real numbers
  – obvious, solved with Calc in a jiffy
  – different implementations may use different notations, but the algorithm is essentially the same
Your Turn (#2) Sudoku

**Rule:** Each row and each column can only contain exactly one of 1, 2, 3 and 4

**Exercise 1:** Solve the 4x4 Sudoku on the left

**Exercise 2:**
Assume that you need to teach a dumb computer to solve Sudoku. You need to make the instructions very explicit. Also, the instructions should work for any Sudoku. What are your instructions?
Your Turn (#3, nifty)

• The lecturer will bring 3 items, each with a 13-digit bar code on it
• Need 3 volunteers
• Each volunteer can choose to change one of the 13 digits or leave them unchanged
  – Of course don’t tell us
  – Read out the correct/modified 13-digit bar code
• The lecturer types the numbers onto the Task3 sheet and presses the Validate! button
• The algorithm will then be revealed, a neat party trick to win friends and influence people
Designing Algorithms

- Need clear specification of problem at hand
- Think of all situations that may arise and know what output to expect
- Does this resemble a standard problem (many identified; some broad classes exist)?
- Even if problem appears to be a new one, it can often be attacked by a small number of general strategies
- Once obtained, need to analyse algorithm for memory consumption, speed, etc.
- May need to repeat this a few times
Algorithm Correctness

- Algorithms can be complex and the tasks they solve difficult
- Errors are easily introduced
- Bugs: can be expensive (and not only financially)
- Can reduce incidence of bugs in three ways: *disciplined design, testing* and *proving*

- **Design**: understand the problem, the intended solution and the notation – *you will try to do this*
- **Testing**: executing program on (lots of) test data – *you can do this and must do this*
- **Proving**: certifying program produces correct result on all permissible data (rarely easy, plus errors may be introduced during coding) – *you probably can’t do this*
Basic components of a computer

Central Processing Unit (CPU) / Processor

Bus

Memory

Picture: http://appleinsider.com/articles/10/05/12/apple_a4_processor_in_iphone_prototype_has_256mb_ram
Primitive instructions

• Instructions the processor can execute
  – Fetch from a memory location
  – Add or subtract 2 numbers
  – Compare two numbers
  – ...

Picture: http://appleinsider.com/articles/10/05/12/apple_a4_processor_in_iphone_prototype_has_256mb_ram
Amazing building blocks

- With a set of well chosen primitive instructions, computers can do many many many different tasks
  - Deeper theory in Church-Turing thesis
What are the “alphabets” of computer instructions?

• Machine language
  – Can be executed directly by CPU
  – 1010011110110000
Programming with high-level computer languages

• High-level programming languages
  – Examples: BASIC, Java, Python, C
  – Must be *translated* into machine language, or *interpreted* step-by-step by another program (increasingly common)

• We will be programming using OpenOffice.org Basic (**OO Basic**, or **OOB** if we really need to abbreviate)
  – OO Basic is bundled with OpenOffice
  – Based on Microsoft’s VBA (Visual Basic for Applications)
OpenOffice.org Basic

• OO Basic uses structures inherited from BASIC (Beginners All-purpose Symbolic Instruction Code) via VBA, with many extensions

• Program is usually stored with document, and can interact directly with document data or other apps

• OOB and VBA allow for object-oriented programming (OOP) like C++, Java, etc.
  – OOP helps programmers solve problems by providing a convenient method for problem decomposition
  – This is outside the scope of this course
  – Object model translated into java objects in the back-end (but you don’t need to know that)
Terminology

• **Macros**
  1. Programs that can be *executed* by user action
  2. General name for active content, including new functions used in formulas, user-designed dialogue boxes etc

• **Execute or run**
  - Transfer control to the Basic interpreter, which performs the specified actions, in order
  - Execution can be traced using *breakpoints* and the program single-stepped

• **Code**
  - informal name for program content, hence *coding*

• **Procedure**
  1. A subprogram or *macro* (sense 1 above)
  2. A function that can be used in a formula
File Formats

• No separate format for OpenOffice documents with added Basic content

• Microsoft product formats have changed several times
  – Common format *xls* derives from Excel 97 (and is still usable), optionally includes VBA
  – Excel 2007 introduced new formats, still current
    • *xlsx* workbook *without stored VBA*
    • *xlsm* workbook with VBA

• OpenOffice can load a Microsoft document that contains VBA, *but*
  – The code is only partly compatible, so
  – VBA is turned into a non-functioning comment, though it’s recoverable
How the program code is organised

- Program content is stored in **modules**, which are grouped into **libraries**
  - Every document has a predefined library called **Standard**
  - **MyMacros** is stored with your OpenOffice installation, it has a **Standard** library too
- Macro organizer gives an explorer-like view
  - Tools – Macros – Organize Macros – OpenOffice Basic
Macro Manager

- You use the manager to
  - View libraries and modules
  - Run or edit a macro
  - Start the Organizer
Macro Organizer

- You use the organizer to
  - Add modules to a library (also in the editor)
  - Create, import or export a library
What you can do with OOB programs

• At the simplest level, you can...
  – Prompt the user for input
  – Process data values
  – Report via the MsgBox dialogue
  – Write functions that can be used in formulas

• With a library to be supplied, you can...
  – Display intermediate values on a log sheet
  – Make decisions based on calculated values
  – Inspect and change cells on the active sheet using row and column coordinates
  – Apply custom algorithms that use sheet data

• With more practice and experience, you could...
  – Interact with the user via custom dialogues
  – Create and animate drawings on screen
During execution, shows values of selected variables

Default code (change and extend)

* As you can run and monitor programs in the editor, it’s also known as an Integrated Development Environment (IDE)

Editing workspace

Current library

Modules in current library
Creating a Sample Program

• Demo only, until we can start using sheet data
  - Select Tools – Macros – Organize Macros – OpenOffice Basic... from menu
  - Press Organizer... on the dialogue
  - Find the document, click + and select Standard
  - Press the New button, default module name is OK
  - Module is added to the list, press Edit
  - Change Sub Main to Sub Golden()
  - Full listing overleaf (lecturer may copy to save time)
First Program (Golden ratio calculator)

' Demonstrates input -> processing -> display with a simple
calculator for objects whose dimensions are in the golden ratio
Option Explicit

Sub Golden()
    Dim width As Double
    Dim goldenRatio As Double  ' or phi
    Dim htPortrait As Double
    Dim htLandscape As Double

    goldenRatio = (1 + Sqr(5))/2

    ' Read the width of an object from the user
    width = InputBox("What is the object width? ","

    htPortrait = width * goldenRatio
    htLandscape = width / goldenRatio

    ' Construct the message using & to glue the parts together
    MsgBox "width = " & width & ", portrait height = " & htPortrait & 

End Sub
First Program (Part 1 of 5)

• We will complete the First Program bit by bit

• Note:
  – It’s a good software development habit to do a small part and then test to see whether it is working. When it’s working, write the next small part of code and test.
  – A **poor** habit is to write a lot of lines of code and then test

```vba
Sub Golden()
    width = InputBox("What is the object width? ", "Golden ratio")
    MsgBox "width = " & width
End Sub
```

*Statements are steps to be executed in turn*
Program Execution

• For this example, we can run it in place
• press F5 (always runs the first procedure in the current module) or press the run button
• InputBox is a quick way of getting value into the program, and MsgBox of showing results:

![Image of InputBox dialog box]

• No fancy stuff like number formatting or line breaks
• Application pauses until dialogue box is dismissed
Program Components

• Subprogram (can be executed by user)
  – between Sub name() and End Sub
  – OO Basic procedures are either subprograms or functions

• Assignment (variable = newvalue)
  – fundamental programming operation, note the order and the operator (= acts like a left arrow \(\Leftarrow\))
Program Components

• **InputBox**
  – Built-in OO Basic procedure to display a prompt and receive a response (string, but convertible to a number)

• **MsgBox**
  – Built-in OO Basic procedure to display something

• "..." are literal strings
  – used for displaying text of some kind

• **&** operator concatenates (joins) strings
  – same notation as used in formulas
Identifiers

Words like *width* in the example program are called **identifiers**

- Identifiers are used for names of procedures, variables, and properties
- Identifiers are sequences of letters (a-z, A-Z), digits (0-9) and underscores (_)
- Identifier can only begin with a letter
- Examples of valid identifiers

```
Module1  x42  temp  blnFound  y_origin
```

**Quiz:** Which of the following identifiers are valid?

```
day 2day dayOfTheWeek day2 $24 see-saw
```
Reserved Words

- Although *day* is a valid identifier, there are problems in using it, because:

- Words like *Sub, End, Dim* are known as **reserved words** or keywords in Basic (same in OOB and VBA)

- You cannot use them as variable names, procedure names, etc.

- Standard procedure names like *MsgBox, day(), month(), year()* etc. are not reserved but avoid them to prevent ambiguity

- Editor highlights reserved words in blue
Rules for choosing identifiers

• Rule 1: Must be valid
• Rule 2: Avoid reserved words

• The program will run if it doesn’t violate Rules 1 and 2

• Rule 3: Choose meaningful identifiers
**Identifier Conventions**

- Identifier conventions have been devised to make programs more readable
  - Use title case for procedure names
    - `FindEmptyCell`  `IsNumeric`  `ToString`
  - Use meaningful variable names, title case with initial lower case, or underscore if capitals would be inappropriate
    - `temperature`  `numCount`  `pressurePa`
    - `mass_in_kg`  `isWithinNormalRange`
  - Other conventions use a prefix* indicating type
  - OK to use short names for minor or short-lived data

* the only instance of this used in ENGG1811 is `o` for object, such as `oSheet` and `oCell`, covered in week 6
First Program (Part 2 of 5)

' Demonstrates input -> processing -> display with a simple  
' calculator for objects whose dimensions are in the golden ratio
Sub Golden()

' Read the width of an object from the user
width = InputBox("What is the object width? ", "Golden ratio")

' Construct the message using & to glue the parts together
MsgBox "width = " & width

End Sub

Comments (begin with single quote ' , ignored)
Comments are important and serve to
explain code, improving its readability, have
no effect on execution

Text inside a Sub (and other structures) should be indented one tab (4 spaces)
Program Style

Programs are both for the computer to run *and* for people to read (you or other people)

- program code is hierarchical (statements are *inside* Sub Golden), so *indent* using tabs (show 4 positions)
  - editor maintains current indent level, which helps
- leave *white space* (between elements and between lines) for clarity
- *continue long lines* with space and underscore _
- *Capitalise* keywords
  - OO Basic does not *require* this, but other Basics (including VBA) do, so we’re going to insist
- add meaningful *comments*
  - *before procedure* explaining purpose, parameters
  - next to important variable *declarations*
  - before or next to important *statements*
First Program (Part 3 of 5)

' Demonstrates input -> processing -> display with a simple calculator for objects whose dimensions are in the golden ratio

Sub Golden()
    Dim width As Double

    ' Read the width of an object from the user
    width = InputBox("What is the object width? ", "Golden ratio")

    ' Construct the message using & to glue the parts together
    MsgBox "width = " & width

End Sub

Variable declarations (named locations to hold values)

Statements are steps to be executed in turn

Text inside a Sub (and other structures) should be *indented one tab* (4 spaces)
Variables

• Variables (Dim name As type)
  – names locations that can be used in calculations

• Variables store values for calculation and later use
  – These values are actually stored in the computer’s memory

• Variables need to be declared before use with the keyword Dim

• Each variable has a data type describing the range of valid values

• Variable names are identifiers (see earlier rules for valid identifier names)
Variables

- Dim var1 As datatype, var2 As datatype, ...
- Dim intX As Long, intY As Long
  - Declares two variables: intX and intY
  - Their data type is Long integer (i.e., whole numbers); these variables can be assigned integer values of either sign, but only up to a limit
- Dim areaPolygon As Double
  - Double = real number approximation using double precision (about 16 significant figures)
- Dim userName As String
  - Declares one variable – userName
  - Data type is String – a sequence of characters
Data Types

- Each variable must have an associated data type
- The data type determines what values can be assigned to variables
- Also determines the amount of memory required to store value of variable
- Data types are important because they allow the compiler* to check for errors in program
- Program also uses data types to determine how to convert a value of one type to another (e.g., an integer to a string)

* The OOB compiler defers most checks until run-time
## OO Basic Primitive Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range of Values Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Boolean</strong></td>
<td>True, False</td>
</tr>
<tr>
<td>Byte</td>
<td>0-255</td>
</tr>
<tr>
<td>Date</td>
<td>Dates and times</td>
</tr>
<tr>
<td>Integer</td>
<td>Whole numbers, -32768 to 32767</td>
</tr>
<tr>
<td><strong>Long</strong></td>
<td>Large integers, +/- 2 billion or so</td>
</tr>
<tr>
<td>Single</td>
<td>Floating point (real numbers, ~ 7 dec digits)</td>
</tr>
<tr>
<td><strong>Double</strong></td>
<td>Higher precision floating point (~ 16 dec digits)</td>
</tr>
<tr>
<td>Object</td>
<td>Generic structured data type</td>
</tr>
<tr>
<td>String</td>
<td>Sequence of characters, variable length</td>
</tr>
<tr>
<td>Currency</td>
<td>Monetary value with up to 4 dec places</td>
</tr>
<tr>
<td><strong>Variant</strong></td>
<td>Dynamic data type (used in special cases)</td>
</tr>
</tbody>
</table>

Use Long for integral counting purposes, and Double for real-number arithmetic.
Illustrating data types and overflow error

• Computers use binary numbers but let us use two imaginary data types for illustration
  – TwoDigits
    • Can store any integer from 0 to 99
  – FourDigits
    • Can store any integer from 0 to 9999
  – Need some volunteers …

• Overflow error
  – Example: Integer data type cannot store integers greater than 32767. If you try to do that, it will result in an overflow error.
Assigning Values to Variables

- A variable can be assigned a value using the assignment operator =
  
  \[ \text{var} = \text{expression} \]
  
  - \text{expression} is evaluated and the result stored in the location named by the variable \text{var}
  - Replaces any previous value

- Examples:
  
  total = 2 + 3  
  areaCircle = 2*PI*radius  
  greeting = "Hello World!"  
  numYing = numYang  
  correct = (total = 5)

  (last one is a comparison assigning True or False)
Assigning Values to Variables

- A variable can be assigned a value using the assignment operator =
  
  \[ var = expression \]
  
  - expression is evaluated and the result stored in the location named by the variable var
  - Replaces any previous value

- The order matters: assign the value on the right to the variable on the left
  
  - ✔ width = 5
  - ✗ 5 = width

Note the order: destination = source
Constant Definitions

- Fixed or **constant** values are often required at several places in a program
- By giving a name to the constant...
  - The reader understands what the value **means**
    - for example, only hard-core physicists would recognise $1.3806503 \times 10^{-23}$ in a calculation (it’s Boltzmann’s constant)
  - The value could be **changed** in one place later if new conditions apply (limits or resource requirements)
- Name format convention: ALL_CAPS

```plaintext
Const PI = 3.141592653589793  ' fundamental value
Const BOLTZ = 1.3806503e-23  ' units are J/K
Const DAYS_IN_LEAP_YEAR = 366
Const MAX_SHEETS = 16  ' some limit
Const DEBUGGING = True  ' controls output
Const VERSION_CODE = "V1.0 beta"  ' info
```
First Program (Part 4 of 5)

' Demonstrates input -> processing -> display with a simple
' calculator for objects whose dimensions are in the golden ratio
Option Explicit

Sub Golden()
    Dim width As Double

    ' Read the width of an object from the user
    width = InputBox("What is the object width? ", "Golden ratio")

    ' Construct the message using & to glue the parts together
    MsgBox "width = " & width
End Sub
' Demonstrates input -> processing -> display with a simple
' calculator for objects whose dimensions are in the golden ratio
Option Explicit

Sub Golden()
    Dim width As Double
    Dim goldenRatio As Double ' or phi
    Dim htPortrait As Double
    Dim htLandscape As Double

    goldenRatio = (1 + Sqr(5))/2

    ' Read the width of an object from the user
    width = InputBox("What is the object width? ", "Golden ratio")

    htPortrait = width * goldenRatio
    htLandscape = width / goldenRatio

    ' Construct the message using & to glue the parts together
    MsgBox "width = " & width _
        & ", portrait height = " & htPortrait _
        & ", landscape height = " & htLandscape

End Sub
Formulas vs Programs

- Spreadsheet formulas are **functional**
  - specify what the answer should be as a single large expression
  - if too complex, intermediate values have to be stored in cells
- OO Basic statements are **procedural**
  - each one is executed in turn
  - all storage locations are *explicitly named*
  - location values can be *updated*

\[ = \text{E2} + \text{F4} \]
\[ \text{num1} = 12 \]
\[ \text{num2} = -3 \]
\[ \text{total} = \text{num1} + \text{num2} \]
Don’t interpret assignment as equals to

We will step through the program UnderstandAssignments()
Note: only part of the program is shown below

Sub UnderstandAssignments()
    Dim x As Integer

    x = 5
    MsgBox "(After x = 5) x = " & x

    x = x + 2
    MsgBox "(After x = x + 2) x = " & x
'Explanation: Starting from the RHS of the assignment statement.
'Take the current value of x (= 5), add 2 to it (which gives 7) and
'assign the result to x. After the assignment statement, x is 7
Arithmetic Expressions

- Used to perform numeric calculations (real or integer)
- Can comprise
  - Literal constants (152, -3, 12.75, 1.39e7)
  - Named constants (PI, MAX, NUM_SHEETS)
  - Numeric variables (x, numDataItems)
  - Arithmetic operators: +, -, *, \, /, Mod, ^
  - Parentheses: ( )

Integer division

Real division

Remainder or modulus
# Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition or unary positive</td>
</tr>
<tr>
<td>–</td>
<td>Subtraction or unary negative</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>\</td>
<td>Integer division (fraction discarded)</td>
</tr>
<tr>
<td>/</td>
<td>Floating point division</td>
</tr>
<tr>
<td>Mod</td>
<td>Integer modulus (remainder)</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation (power)</td>
</tr>
</tbody>
</table>
Examples of Expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 + 2 * 3 - 4</td>
<td>3 (not 5)</td>
</tr>
<tr>
<td>5 / 2</td>
<td>2.5</td>
</tr>
<tr>
<td>5 \ 2</td>
<td>2</td>
</tr>
<tr>
<td>14 Mod 5</td>
<td>4</td>
</tr>
<tr>
<td>2 ^ 3</td>
<td>8</td>
</tr>
</tbody>
</table>

sum + 1
curPrincipal * (1 + interestRate) ^ numYears
(a + b) Mod 10
(R1 * R2) / (R1 + R2)
a*x^2 + b*x + c
Precedence

- When evaluating arithmetic expressions, order of evaluating operations determined by *precedence*

<table>
<thead>
<tr>
<th>Operator</th>
<th>Higher precedence</th>
<th>Lower precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* /</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ -</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- You can look this up when needed, supplied in exams too
Evaluating Expressions – Rules of Precedence

• When evaluating expressions, operations of higher precedence are performed before those of lower precedence
  \[ 2 + 3 \times 4 = 2 + (3 \times 4) = 14 \]

• Otherwise operations performed from left to right
  \[ 2 ^ 3 ^ 4 = (2 ^ 3)^4 = 4096 \]
  \[ 10 + 2 - 3 = 9 \]

• Use parentheses if in any doubt
Quiz

• What is \(-2^2\) in OOB?
  (a) 4           \((-2)^2\)
  (b) -4          \(- (2^2)\)

### Operator

<table>
<thead>
<tr>
<th></th>
<th>Higher precedence</th>
<th>Lower precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ – (unary: sign)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* /</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mod (remainder)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ – (binary: add, subtract)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Important Note:** Different programming languages can use different orders of precedence. Always check or use () when in doubt.

The OOB convention is different from standard maths.
Tracing execution

- As the program is interpreted, it can be paused, resumed and stepped through.
- Set a breakpoint by double clicking in the left margin next to a statement.
- Execution pauses, continue with F8 or use toolbar:
  - Step into (F8), next statement
  - Step over (Shift-F8), treat procedure as single step
  - Step out, step to end of current procedure
- When paused, hover mouse over variable name to see its current value.
- If you’re really keen, set a variable watch.
What you really need to know

- After this last lecture for the week, you must know
  - how to create and edit a new module, and find your way to existing modules
  - how to create and run a subprogram
  - how to declare variables, and what data types mean
  - what assignment statements do
  - how to use MsgBox (and strings, and the & operator)
  - how to trace execution using breakpoints and F8
  - the rules for forming identifiers, and how to name variables
  - program style conventions
  - how to define named constants and naming conventions
  - about arithmetic expressions (including the Mod operator) and precedence

- The lab work after the midterm test will require this!
Summary

• Algorithms express solutions to problems
• Programs implement algorithms
• OO Basic is a particular language with its own way of representing data and action
• OO Basic is bundled with OpenOffice
• Use the built-in editor (IDE) to edit and test
• Programming concepts
  – procedures for grouping code
  – variables, types, constants
  – assignment (change value of variables)
  – arithmetic expressions for evaluation