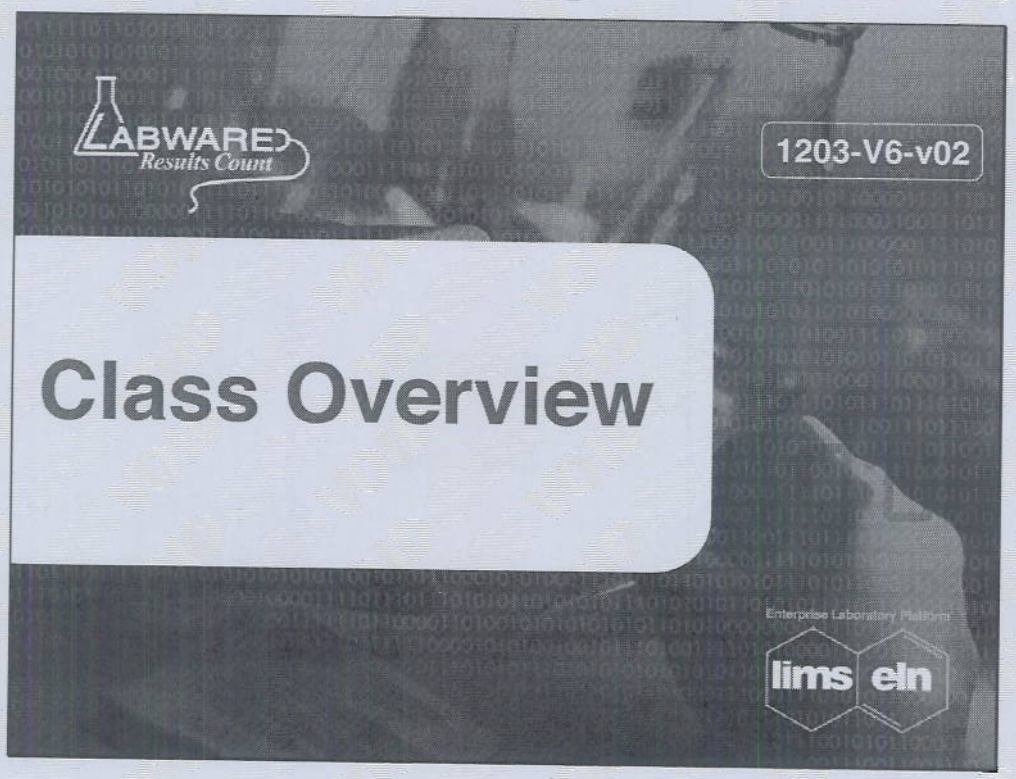
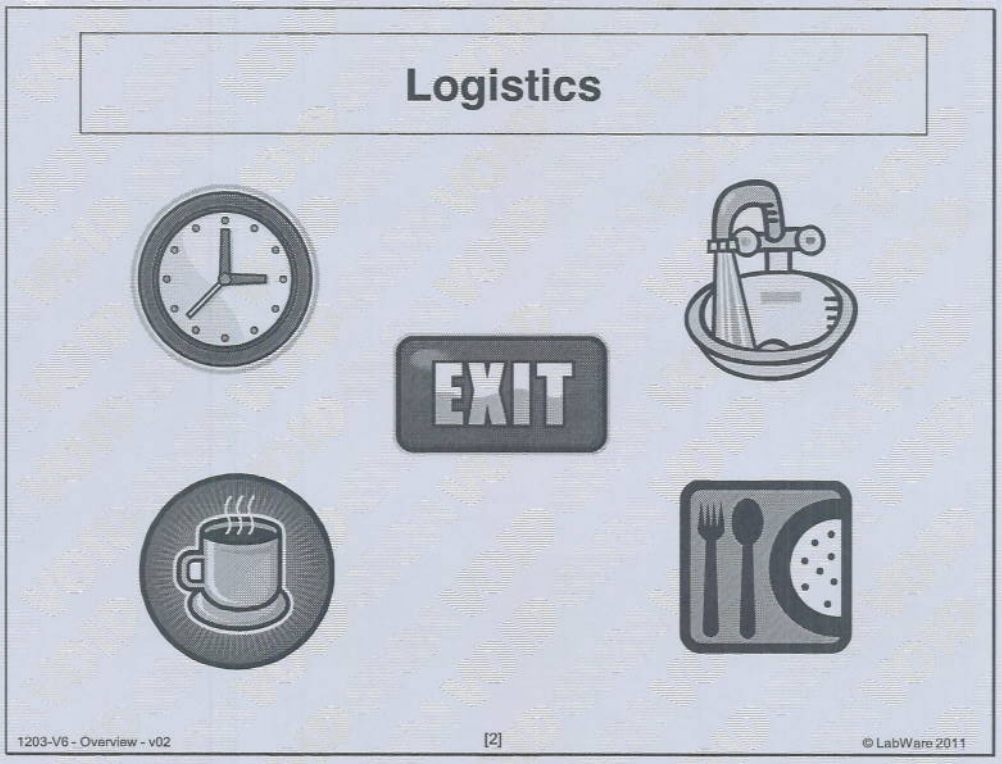
Ajit Pillai

# Table of Contents




>>Duration: 10 minutes




- Domestic arrangements
- Restrooms

## Introductions

**Company**

**Department / Functional Area**

**Personal background**

**LIMS experience**

**Objectives**

1203-V6 - Overview - v02 [3] © LabWare 2011

- Lab activity: QA, R&D, commercial?
- Personal: IT/database/programming experience?
- LIMS experience: experience or role

## LIMS Basic Training: Desired Learning Outcomes

1. Understand simple programming
2. Create subroutines using functions and statements within LabWare LIMS
3. Create Macros, Automation Scripts and Event Triggers using LabWare LIMS Basic functions
4. Recognise how the use of LabWare LIMS Basic may be useful in your functional area

1203-V6 - Overview - v02 [4] © LabWare 2011

# Agenda

Day 1: Introduction to LIMS Basic
- What is LIMS Basic, elementary concepts
- Simple functions and statements

Using Functions and Statements in LIMS Basic

Day 2: Running SQL from LIMS Basic
Macros

Day 3: Event Triggers, Automation Scripts, Calculations, Template Formulas, Format Calculations, Id Configuration

## Training Room

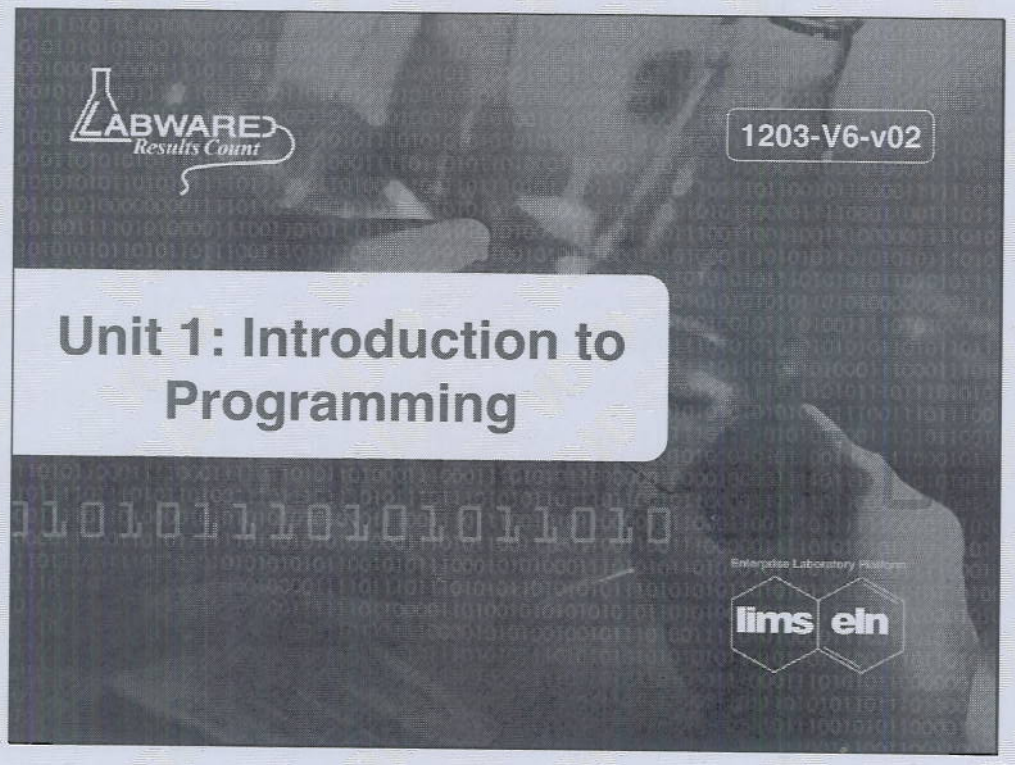
- This is a "hands-on" course.
- Don't hesitate to explore and experiment with the interfaces as they are being explained to you. You won't break anything!
- If you are "stuck" please inform trainer for assistance.
- Ask any questions you like at any time.
- Please ensure cell phones are off or on vibrate.

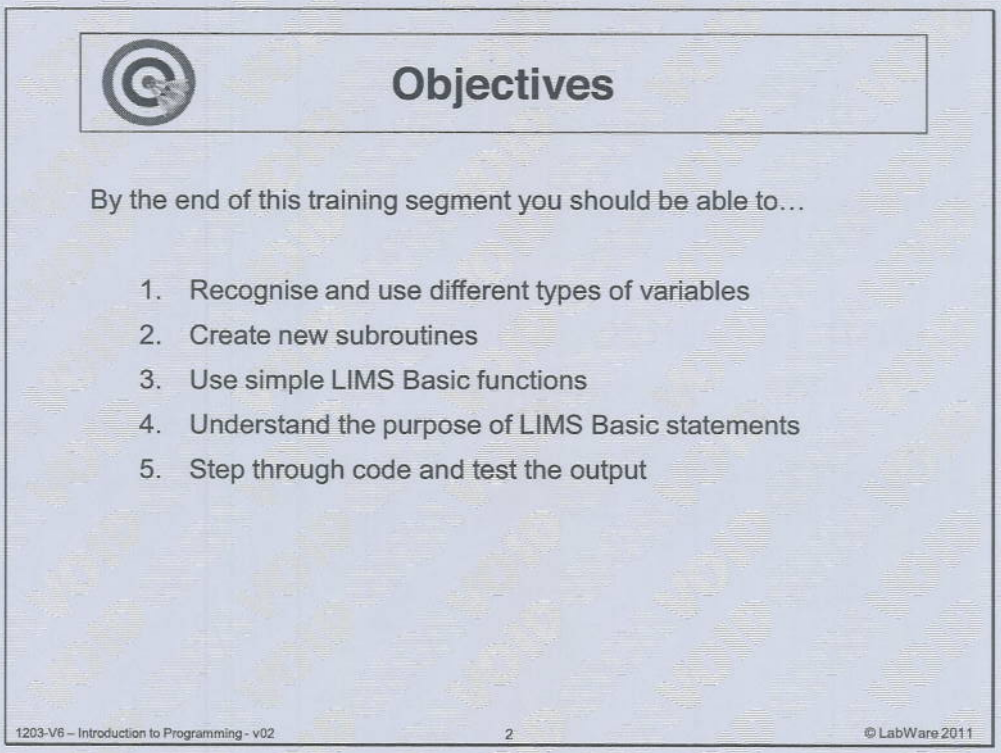
# Course Overview

- Designed with both novice and experienced programmers in mind
- Pre-requisites:
  - LIMS Admin 1 course
  - A good all-around knowledge of LabWare LIMS configuration
- LIMS Basic functions
  - There are several hundred LIMS Basic functions (developed and extended on an on-going basis)
  - Not all of the functions will be covered in this course
  - Additional resources / references:
    - On-line Help Files
    - LIMS Basic User Manual
    - LabWare White Paper: LIMS Basic Guidelines and Object Naming conventions (LabWare Support web page)

# Exercises

- At the beginning of each exercise you will find a list of the functions and statements required to complete the exercise
- The examples used in the exercises may not be the only way to achieve the same outcome
- Solutions to the exercises are provided at the end of the training manual


LABWARE
Results Count
1203-V6-v02
Unit 1: Introduction to Programming
Enterprise Laboratory Platform
lims eln

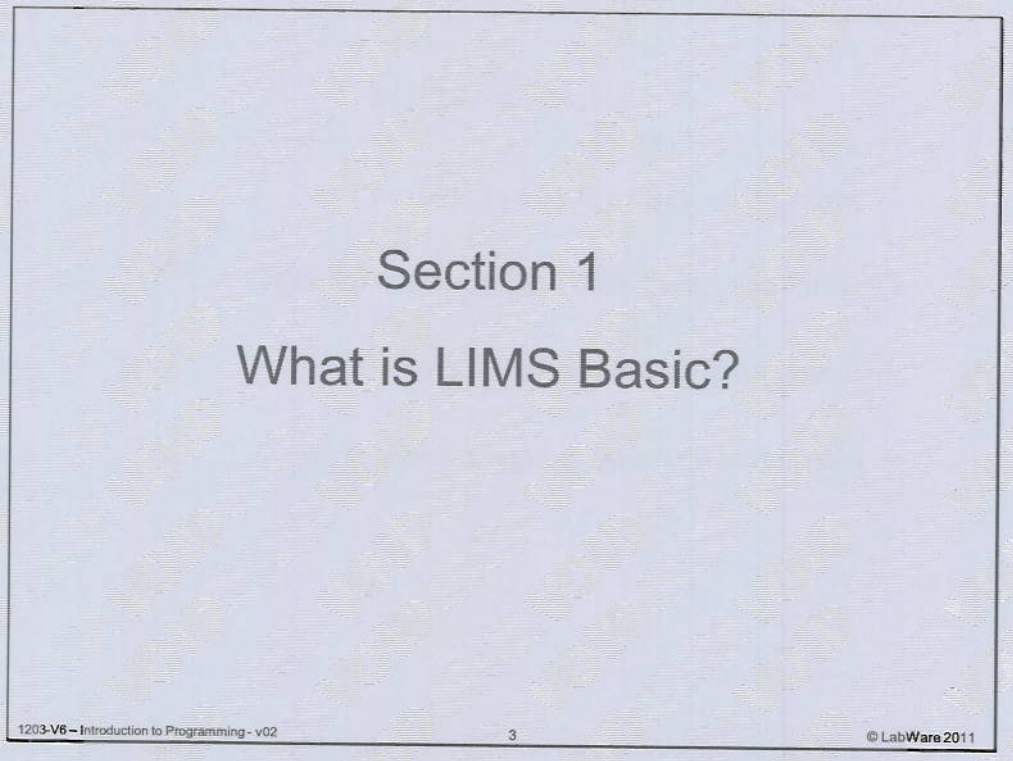


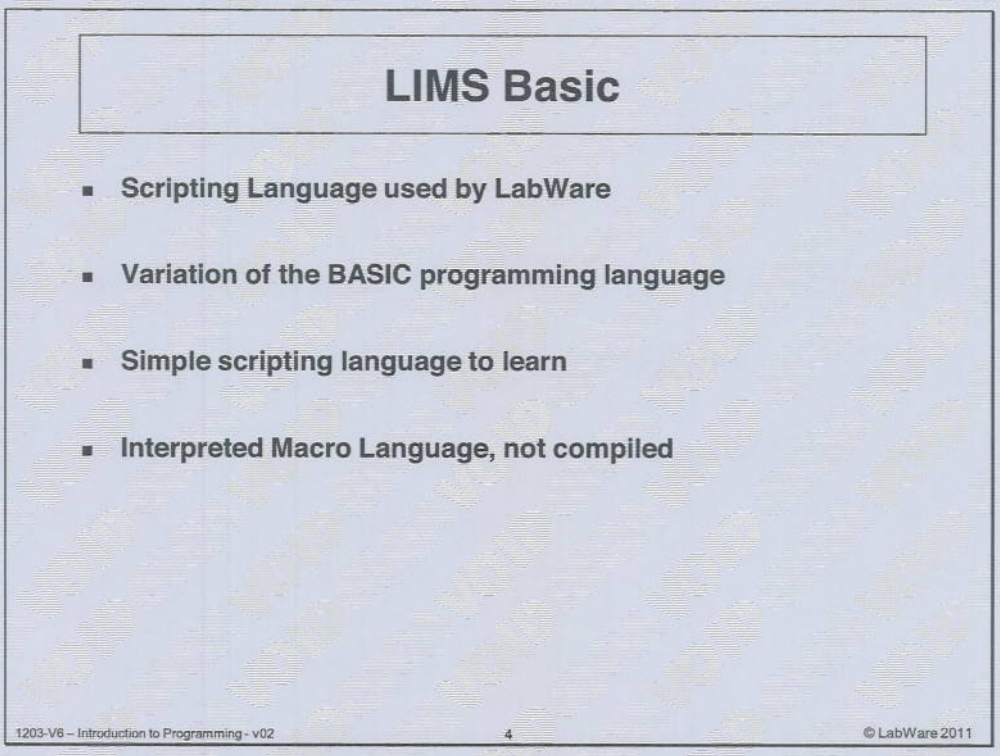

- This section is an introduction to LIMS Basic using simple variables, statements and functions.

New concepts
- LIMS Basic
- Variables
- Subroutines
- Functions and Statements
- LIMS Basic debugger

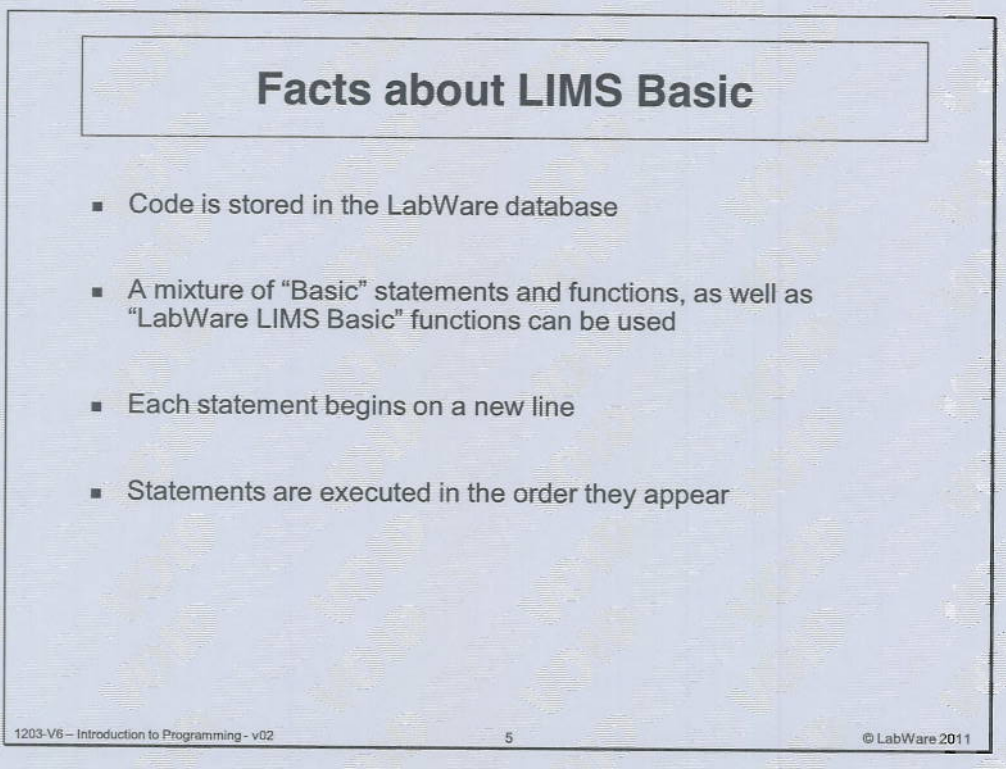
New Functions
- Chr()
- Date()
- InputBox()
- MsgBox()
- Time()

New Statements
- DO LOOP
- FOR NEXT
- IF THEN
- SELECTCASE
- WHILE WEND


Section 1
What is LIMS Basic?
1203-V6 – Introduction to Programming - v02
3
© LabWare 2011




- LIMS Basic is a Scripting Language that is available in LabWare to allow users to introduce logic into the configuration of their system. For example:
  - You could configure an Analysis rule that uses LIMS Basic to automatically create an investigation and alert the QC Manager when the E. coli bug is found in a Microbiology sample.
  - You could also configure an extra menu in the Project Manager that allows the user to re-allocate samples to another project
- BASIC is the acronym for **B**eginner's **A**ll Purpose **S**ymbolic **I**nstruction **C**ode. The BASIC programming language uses "ordinary English" to perform processor tasks. BASIC was introduced in 1964 and many variations of the language exist today.
- LIMS Basic is a variation of BASIC and is very simple to learn.
- LIMS Basic does not require variable typing, declaring or dimensioning as is the case with various other programming languages.
- LIMS Basic code is not compiled. The basic interpreter reads the code and translates the code in to actions. The code can be written and immediately tested.




- All LIMS Basic code is stored in the LabWare Database (not in a separate executable)
- There are functions and statements that are similar to "BASIC" type statements, as well as LabWare LIMS specific functions
- Each statement must begin on a new line
- Statements are executed in the order that they appear (top to bottom)

# Section 2

# Elementary Concepts

## What is a variable?

- Instructions to a computer are called a PROGRAM
- VARIABLES are placeholders used to store values
- Example of a program to add two numbers:

  *line1* a = 2

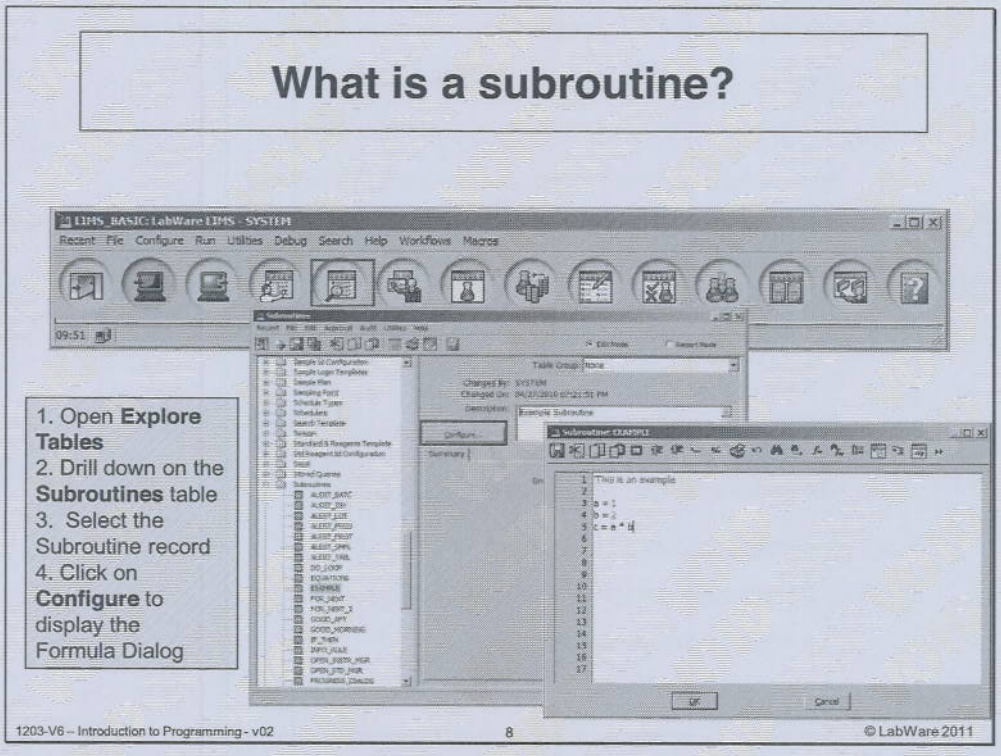
  *line 2* b = 5

  *line 3* c = a + b

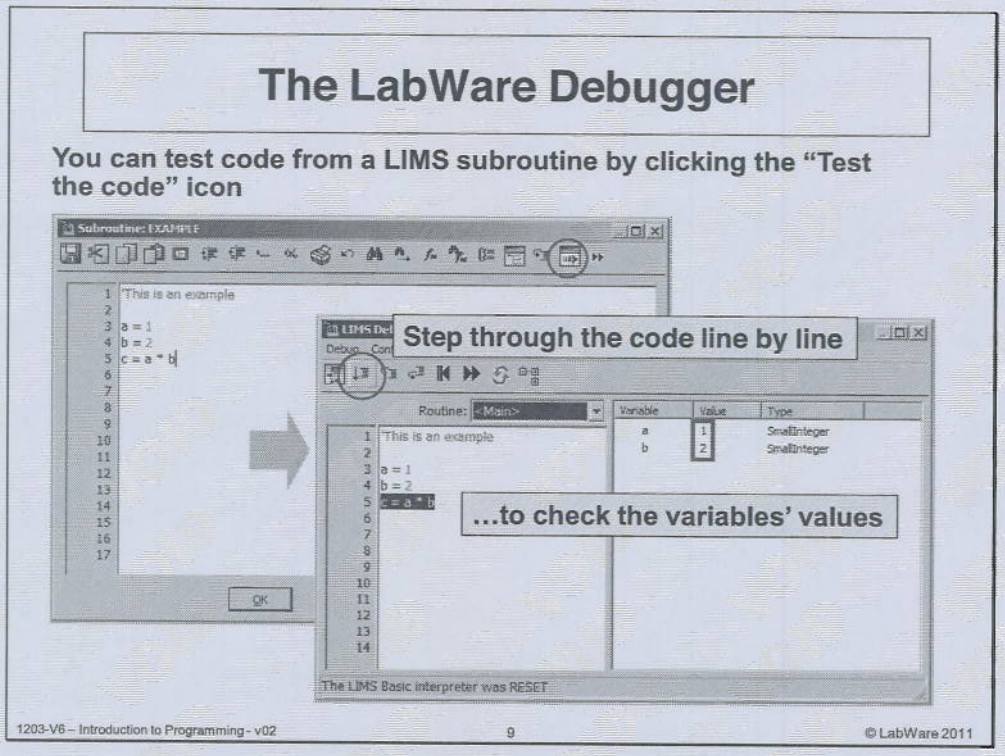
- The letters "a", "b" and "c" are **VARIABLES**
- A variable can hold a value:
  - Specifically assigned to the variable
  - As a result of processing data

1203-V6 – Introduction to Programming - v02 7 © LabWare 2011

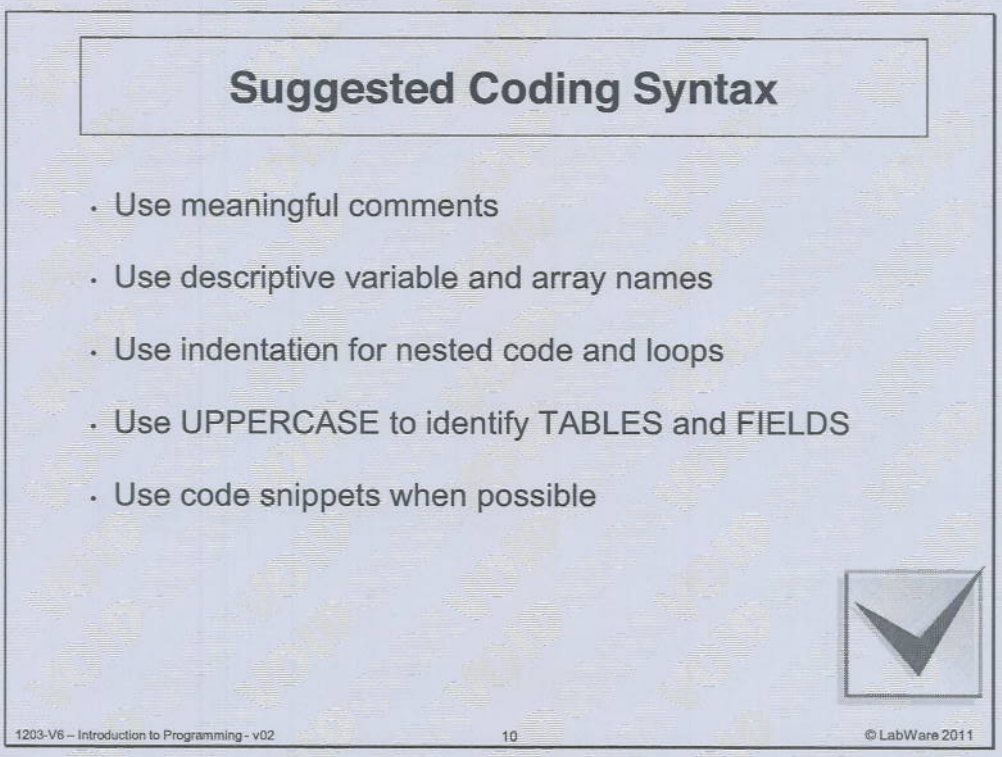
- "Programming" allows a machine to perform tasks and calculations that people can do but in a more efficient manner.
- Before the computer can perform the instructions or do the calculations it first has to be told exactly what to do. The instructions to the computer are called the PROGRAM.
- What we see, read or type for the program is simply for readability for ourselves, reviewers and users. The machine or computer only understands machine-readable code, such as binary (0's and 1's). Fortunately we do not have to do this translation ourselves; the programming language does this for us.
- The calculation listed in the example is a simple calculation to add two numbers together. We can easily define the answer as "7". The computer has to store each part of the sum within variables.
- A variable is assigned a value specifically or as a result of processing data.
- Variable values may be strings, numbers, arrays, date-times, times or dates.
- You may need to convert a variable's value type, e.g. from a number to a string to use the value in the Msgbox function.
- Variable names must start with a character. Variable names may not be a reserved word, e.g. USER, EMPTY, ERROR




- The easiest place to start writing or testing LIMS Basic code is in a **subroutine**.
- New Subroutines can be created in the Subroutines table.
- Subroutines are discreet pieces of code which can be called from various events within LabWare.




- Click on the **Test the Code** button on the toolbar. This will open the LabWare Debugger window.
- To execute the code line by line, click on the **Step** button.
- The first line of code above is not executed because it is a **comment**.
- Comments are added to code for clarification and documentation purposes.
- To "comment out" a line of code, simply make the line start with an apostrophe (‘). Comments can also be added at the end of a line.
- Comments can be added to a selection of code by highlighting the code and selecting the comment icon within the subroutine.




- The LabWare White Paper called "LIMS Basic Guidelines and Object Naming Conventions" provides best practices when writing LIMS Basic.
  - While LIMS Basic is not case sensitive it is suggested that when identifying variables multi-case names are used for easier readability, e.g. sampleNumber or productName
  - Tables and fields that are referenced in SQL calls or code are suggested to be in all CAPS
- It is important to add comments to the code, not only to describe what the code is meant to do, but to also describe the reason it was built to perform the way it was established.
- Meaningful comments are useful for reviewers, other programmers, and the support team to understand the purpose of the written code.
- **Note: Not all lines of code need to be commented, just enough to add meaning.**
- Code Snippets are defined block of LIMS Basic code (templates) that have been or can be created to assist with proper coding technique or to simplify coding.
  - Code snippets are stored in the Code Snippets table.
  - A block of code can be saved as a Code Snippet from the Formula Dialog window.
  - A Code Snippet can be inserted into a Subroutine or Formula Dialog from the Formula window.

# Exercise LB01-01: Variables

In this exercise, you will create a subroutine with a simple calculation using variables.

1) Create a subroutine called NUMBERS.

2) Write a few lines of code which take the two values 7 and 9 and multiply them by each other.

3) Save the subroutine.

4) Test the LIMS Basic code by stepping though the code using the LIMS debugger. Notice the variable values in the right hand pane of the debugger window.

5) If time allows, add comments and code header to your subroutine.

## Variable Types

- LIMS Basic variables can have any name, apart from a few reserved words listed in the LIMS Basic manual
- Here are some examples of values a variable might contain:

| | |
|---|---|
| **message = "Hello, this is a string"** | *= text (string)* |
| **a = 2** | *= small integer* |
| **mass = 39.2932** | *= floating point (real number)* |

- Open the subroutine VAR_TYPES to test this code.

1203-V6 – Introduction to Programming - v02 12 © LabWare 2011

- In some programming languages:
  - The type of variable has to be specifically defined before it can be used.
  - Variables have to have certain types of names depending on whether they are to store integers, floating point decimals, text, etc.
- **In LIMS Basic**:
  - The variables can have any name, apart from a few reserved words listed in the LIMS Basic Manual.
  - A variable does not have to be defined before it can be used
- As we saw in the previous section, variables can also be used to perform calculations. For example: diff = end – start
  - Uses the two variables called "start" and "end" to store data or values.
  - It subtracts the variable values from each other and the answer is stored in the variable "diff".
- Although the "diff = end – start" code may look like an algebraic formula, it is not.

## Variable Types

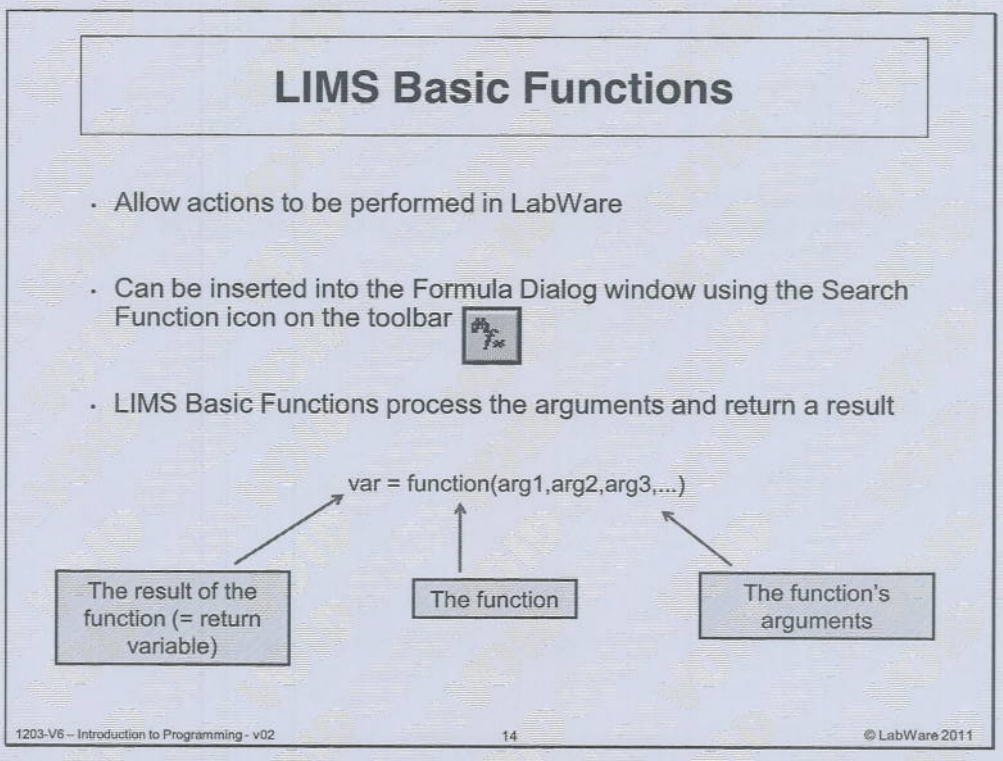
- Although code may look like algebraic equations, programming languages do not follow algebraic behavior. Evaluate the following example:

| | | Value of x |
|---|---|---|
| *line1* | **x = 0** | **0** |
| *line 2* | **x = x + 5** | **5** |
| *line 3* | **x = x + 5** | **10** |
| *line 4* | **x = x + 5** | **15** |

- Open the subroutine EQUATIONS to test this code.

1203-V6 – Introduction to Programming - v02 13 © LabWare 2011

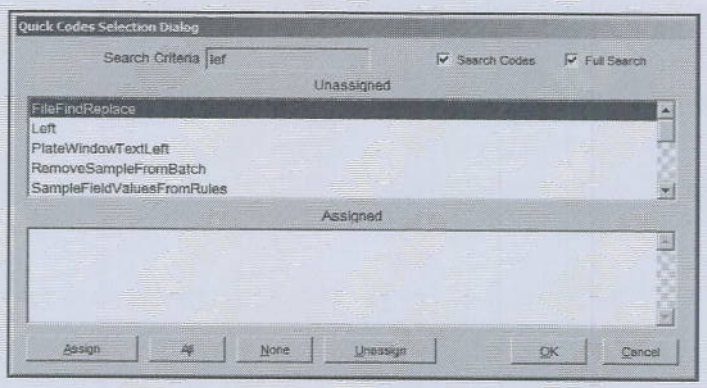
- Mathematically or algebraically speaking x would not be redefined in each line of this example. The value of x is always "0".
- However in programming languages, the value of x is redefined in each line as shown.




- A LIMS Basic function is a pre-defined block of code that will perform a specific LabWare action.
- Think of all the things you can do interactively, e.g. Log a Sample, Add a Test, Enter a Result. There are literally hundreds of LIMS Basic functions. They are also known as **commands**.
- Most often LIMS Basic functions have arguments, which is the information contained in the parentheses following the function name. The values in the arguments can be used to perform the actions.
- Some LIMS Basic functions, for example arithmetic functions, are common to all BASIC languages. There are also many functions that are specific to the LabWare environment.
- **TIP: It is strongly recommended that you always insert a LIMS Basic function. This will prevent syntax errors and reduce debugging.**

## Inserting a LIMS Basic Function

- The Search Function icon can be used to open the Quick Codes Select Dialog 




1203-V6 – Introduction to Programming - v02 15 © LabWare 2011

- >>> Describe the use of the Quick Codes Selection Dialog
- >>> Always insert, rather than type

## Arguments in a LIMS Basic Function

- Arguments are variables holding specific data values that will be used when the LIMS Basic function is executed
- Example: The Left function can be used to extract a number of characters from a string

  part = Left(string,num)
- The arguments can be defined as follows:

  string = "LabWare"

  num = 3
- The function will return the 3 left most characters of the string to the variable with the name part

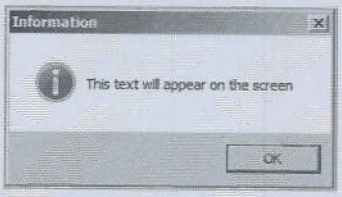
  part = "Lab"

1203-V6 – Introduction to Programming - v02 16 © LabWare 2011

- In this example the Left LIMS Basic function is used: part = Left(string,num)
- **part** is the return variable.
- **Left** is the name of the function.
- **String** and **num** are the function's arguments; which are **separated** by a comma.
- The function processes the arguments and gives you an answer (value) in return.
- Create a new Subroutine called TEST to configure the code in the example.
- Test the code in the Subroutine.
  - Notice the value of the variable **part** at the end of the code.
  - What happens if you change the value of the variable **num** to 4?

## Function Demo: Msgbox()

- Message boxes can be used to display "messages" to Users.
- Syntax: MsgBox(string)
  - **String** is the variable
- Example:
  string = "This text will appear on the screen"
  MsgBox(string)




1203-V6 – Introduction to Programming - v02 17 © LabWare 2011

- The Message Box function: MsgBox(string)
- **String** is argument in the function.
- The value of **string** is typed within double quotes to indicate that it is a string (text).
- Message boxes are not only used to display a message to the user. As you will see later they can also be useful while building your code to detect errors.

## Exercise LB01-02: Message Box

In this exercise, you will create a subroutine with a simple string variable and you will display it in a message box.

**Functions:**
**MsgBox(string)**
**Statements:**
**None**

1) Create a subroutine called HELLO and insert the Msgbox function.

2) Define the string variable as "Hello!".

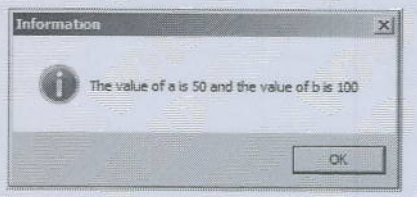
3) Save the Subroutine.

4) Test the code in the subroutine to display the variable's value.

## Concatenating Variables

- Messages to be displayed can be built using a combination of strings and variables
- The ampersand (&) or plus (+) sign is used to concatenate the values together.
- Example:

```
a = 50
b = 100
string = "The value of a is " + a + " and the value of b is " & b
MsgBox(string)
```




1203-V6 – Introduction to Programming - v02 19 © LabWare 2011

- The & and + signs are inter-changeable to concatenate variables together.
- It is a matter of preference to which you use.
- Through-out this training, you will notice that we have used either with no preference.

## Exercise LB01-03: Concatenating Variables

In this exercise, you will configure a subroutine that displays the first name and surname of John Smith in a message box.

**Functions:**
**MsgBox(string)**
**Statements:**
**None**

1) Create a subroutine called USERNAME.
2) First create a string variable with the value "John".
3) Create another string variable with the value "Smith".
4) Create a third variable that concatenates both the first name and surname. Don't forget to insert a space between the two words.
5) Display the third variable in a message box.

## Exercise LB01-04: Concatenating more Variables

In this exercise you will configure a subroutine that calculates the circumference of a circle with a radius of 7 cm. You will display the result to the user in a message box.
Circumference formula: $c = 2\Pi r$ . Note: $\Pi = 3.14$

**Functions:**
**MsgBox(string)**
**Statements:**
**None**

1) Create a subroutine called CIRCLE

2) Define each of the calculation parameters as separate variables.

3) Define a string variable that shows what the result is. Your string should be displayed like this:

**The circumference of the circle is ..... cm**

4) Display the string in a message box.

1203-V8 – Introduction to Programming - v02 21 © LabWare 2011

# Section 3

## Some simple functions

# Function Demo: InputBox()

**InputBox()**

**Purpose:** The InputBox function displays a dialog box that prompts the user for input.

**Syntax:** x = InputBox(*prompt*,*title*,*default*)

**Comments:** *prompt* is displayed in the dialog box.
*title* is optional and is displayed in the title bar of the dialog box.
*default* is default response if no other input is entered.

**Example:**
```
prompt = "What is your name?"
title = "NAME"
default = " "
x = InputBox(prompt,title,default)
```

**Returns:** x contains the information typed by the user.

## Good Programming Hints

1. Insert functions with the Search Function icon  to ensure syntax is correct.

2. Define variable arguments BEFORE the function and NOT inside it.

```
name = InputBox("What is your name?", "NAME", " ")
```

3. Give meaningful names to your variables.

```
x = "What is your name?"
y = "NAME"
z = ""
n = InputBox(x,y,z)
```

```
prompt = "What is your name?
title = "NAME"
default = ""
name = InputBox(prompt,title,default)
```

1203-V6 – Introduction to Programming - v02 24 © LabWare 2011

- Do not try to type functions from memory. Use the Search Function icon to search for and insert a LIMS Basic function. This will help reduce error checking.
- **Spelling matters; copy and past or insert whenever possible! Double click on a word to highlight / select and then perform copy and paste.**
- If you define the arguments inside the function you may save a few lines, BUT your code will be more difficult to debug.
- Use meaningful names for your variables

# Exercise LB01-05: Input Box

In this exercise, you will write a subroutine that prompts the user to enter his first name and age in input boxes. The information will then be processed and displayed in a message box.

**Functions:**
**InputBox()**
**MsgBox()**
**Statements:**
**None**

1) Create a subroutine called GREETING_1.

2) First ask the user to type his name in an input box.

3) Ask him to enter his age in a second input box.

4) Display the result in a sentence like this:

**Hello ....., so you are ... years old. That's interesting!**

## Function Demo: Chr()

**Chr()**

**Purpose:** The CHR function converts an ASCII code to its equivalent character.

**Syntax:** x = CHR(integer)

**Comments:** "integer" is an integer value in the range of 0 to 255. Refer to the table of ASCII characters for values.

**Example 1:**
x = CHR(84)
Returns: x = T

**Example 2:**
```
Message = "This is the first line" + CHR(10)
Message = Message + "This is the second line" + CHR(10)
Message = Message + "This is the third line"
MsgBox(Message)
```

Returns: A message box displaying the following:
This is the first line
This is the second line
This is the third line

1203-V6 – Introduction to Programming - v02 28 © LabWare 2011

- Each character on your computer keyboard has its very own unique numeric code called an ASCII code. For example, the ASCII code for the letter T is 0084. If you know the ASCII code of a character, you can usually insert it in a document by typing Alt+ASCII (Alt_0233 will insert the letter é).
- Even the space bar and the carriage return have their own ASCII code. This allows them to be used in programming, for example a carriage return can be inserted so that it will display a string on a new line in a message box.
- The function for new line is Chr(10) and carriage return is Chr(13).
- ASCII code definitions can be found online.

## Exercise LB01-06: Message Box with multiple lines

In this exercise, you will write a subroutine that prompts the user to enter his first name and surname in input boxes. The information will then be processed and displayed in a multi-line message box.

**Functions:**
**Chr()**
**InputBox()**
**MsgBox()**
**Statements:**
**None**

1) Create a subroutine called USER_INFO.

2) First ask the user to enter his first name in an input box.

3) Ask the user to enter his surname.

4) Create a string variable to display the result in a message box on 2 separate lines. The message box should look like this:

# Function Demo: Date()

**Date()**

**Purpose:** The Date function returns the current system date.

**Syntax:** today = Date()

**Comments:** today is a date variable type.

**Example:** today = Date()

**Returns:** today = 07/22/2000

# Function Demo: Time()

**Time()**

**Purpose:** The Time function returns the current system time.

**Syntax:** now = Time()

**Comments:** now is a time variable type.

**Example:** now = Time()
**Returns:** now = 11:09:45

## Exercise LB01-07: Date() and Time()

In this exercise, you will create a subroutine with date and time variables and you will display them in a message box.

**Functions:**
- **Chr()**
- **Date()**
- **MsgBox()**
- **Time()**

**Statements:**
- **None**

1) Create a subroutine called DATE_INFO.

2) Insert the date() and time() functions.

3) Create a string variable that displays the date and time in a message box like this:


4) Notice the variable types as you test your code in the LIMS Debugger

# Section 4

# Statements

## What is a statement?

- Statements allow the introduction of logical structure within LIMS
- For example:

  IF a condition is true THEN

    do something,

  ELSE

    do something else.

  ENDIF
- Statements can be inserted using the Insert a Statement button 
- Available statements

| | | |
|---|---|---|
| COMMENT | GOSUB | SELECT |
| DO LOOP | IF THEN | SELECT CASE |
| FOR NEXT | RETURN | WHILE WEND |




- LIMS Basic statements give structure to the code. You could say that if the functions were the "flesh" of the code, the statements would be the "bones".
- Statements are lines of LIMS Basic code. Statements can consist of expressions, functions, and variables.
- LIMS Basic requires specific formats for some statements.
- For example, an **IF THEN** statement is a conditional structure. It allows you to introduce logic:
  **IF** a certain condition is true (e.g. my sample is in spec)
  **THEN** do one thing (e.g. auto-approve it)
  **ELSE** do another thing (e.g. send an alert)
  **ENDIF**
- These structures are very useful to automate "routine" decision making processes.
- Most LIMS Basic statements are generic BASIC statements, so if you have previous experience in this, you will find them quite easy to use.
- Once more, if you always insert statements instead of typing them in, you will save yourself a lot of syntax errors.

## IF THEN Statement

**Purpose:** The IF THEN statement allows conditional execution of statements based the evaluation of a logical expression.

**Syntax:**
```
IF (expression) THEN
    statements...
ELSE
    statements...
ENDIF
```

**Comments:** The *expression* must evaluate to True or False.
The full *expression* must be enclosed in parentheses.

**Example:**
```
x = 1
y = 10

IF (x <= y) THEN
    MsgBox(x)
ELSE
    MsgBox(y)
ENDIF
```

Test this code in the IF_THEN subroutine

1203-V6 – Introduction to Programming - v02 33 © LabWare 2011

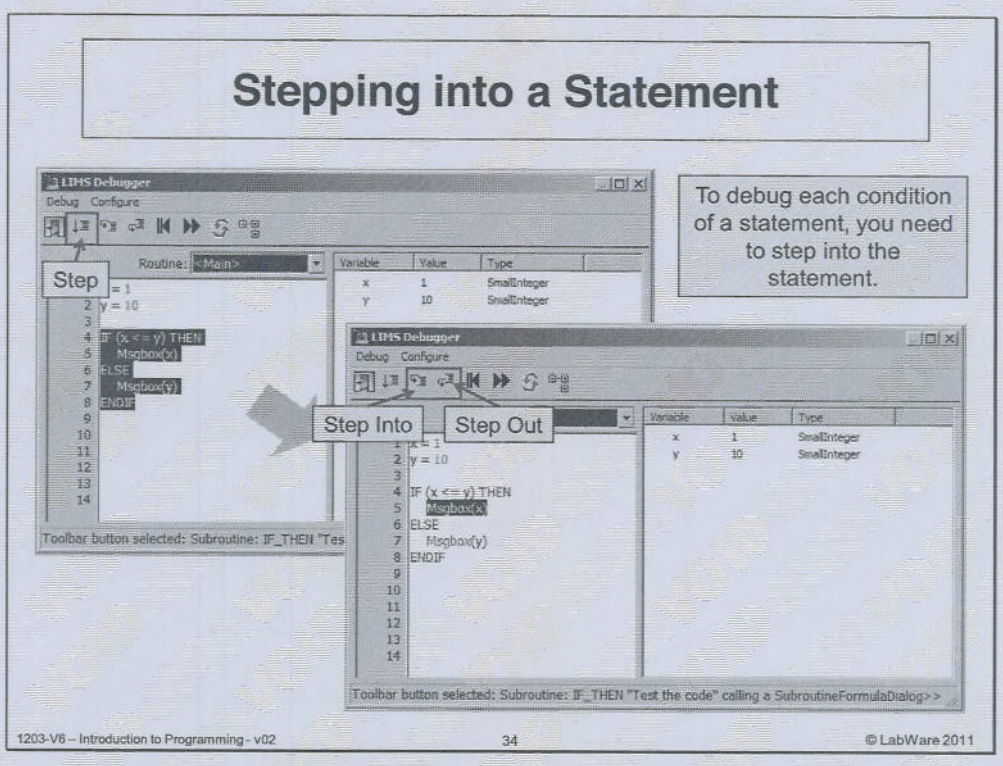
**Optional ELSEIF**

- It is possible to test additional conditions using the ELSEIF.

- For example:

```
x = 1
y = 10

IF (x < y) THEN
    Msgbox(x)
ELSEIF (x = y) THEN
    x = Str(x)
    y = Str(y)
    xy = x + " – " + y
    Msgbox(xy)
ELSE
    Msgbox(y)
ENDIF
```




- When testing or debugging LIMS Basic code the default behavior is not to step into a statement.
- Initially, the debugger highlights the whole statement. The step button will evaluate all conditions and execute the subsequent actions in one go.
- To step into the Statement click on the Step Into button. Once in the statement you can use either the Step or Step Into buttons to step through the code.
- Use the Step Out button to step out of the statement.

## Function Demo: PromptForDateTime()

**Purpose:** The PromptForDateTime function prompts the user for date and time value.

**Syntax:** dt = PromptForDateTime(title, default)

**Comments:** dt is a DateTime data type.

**Example:** dt = PromptForDateTime()

**Returns:** dt = 08/28/2010 01:24:36 PM

1203-V6 – Introduction to Programming - v02 35 © LabWare 2011

- This is an additional function that can be used in place of prompting users with an InputBox().
- Try using this in the next exercise instead of an InputBox(). What other functions might you need to determine if it is morning or afternoon?

## Exercise LB01-08: Finding out if it's morning or afternoon

In this exercise you will work out whether it's morning of afternoon and you will display your finding in a message box.

**Functions:**
**InputBox()**
**MsgBox()**

**Statements:**
**IF THEN**

1) Create a subroutine called GREETING_2
2) First prompt the user to enter a time in a format HH:MM:SS (24 hours format)
3) Now work out whether the entered time is before or after 12:00:00
4) If it's morning, display the message: **Good morning!**

   If it's afternoon, display the message: **Good afternoon!**
5) If time allows, try using the PromptForDateTime()

1203-V6 – Introduction to Programming - v02 36 © LabWare 2011

- PromptForDateTime() will need the use of Left(), Right() and/or Mid()

# SELECT CASE Statement

**Purpose:** The SELECTCASE statement executes one of several statement blocks depending on the value of a variable.

**Syntax:**

```
SELECTCASE variable
CASE comparison list (e.g."A", "B")
    statements...
CASE comparison list (e.g. "C")
    statements...
CASE ELSE
    statements...
ENDSELECT
```

**Example:**

```
x = 3
SELECTCASE x
CASE 2,4,6
    MsgBox("x is even")
CASE 1,3,5
    MsgBox("x is odd")
CASE ELSE
    MsgBox("something else")
ENDSELECT
```

Test this code in the SELECTCASE subroutine

# Function Demo: Right()

**Right()**

**Purpose:** The Right function returns the rightmost number of characters in a string.

**Syntax:** x = Right(*string, num*)

**Comments:** If *num* is greater than the length of the *string*, the entire string is returned. If num is 0, an empty string is returned.

**Example:**
string = "LabWare"
num = 4
x = Right(string, num)

**Returns:** x = "Ware"

## Exercise LB01-09: SELECTCASE

In this exercise you will modify the SELECTCASE subroutine so that the user is prompted for a number to test. The subroutine should work for ANY number, no matter how big.

**Functions:**
**InputBox()**
**MsgBox()**
**Right()**
**Statements:**
**SELECTCASE**

1) Prompt the user for a number
2) Look if that number is odd or even. Tip: look at the last digit of the entered number.
3) If the number is odd, display the message: <entered number> **is odd.**
4) If the number is even, display the message: <entered number> **is even.**

1203-V6 – Introduction to Programming - v02 39 © LabWare 2011

## LOOPS

- Loops execute a section of code a number of times
- There are several types of loops:
  - DO LOOP
  - WHILE WEND
  - FOR NEXT
- The DO LOOP and WHILE WEND loops continue *while* a condition is true
- The FOR NEXT loop continues for a *defined* number of loops.

1203-V6 – Introduction to Programming - v02 40 © LabWare 2011

- Loops allow the repetition of a section of code for a number of times. For example, you might want to log several samples at once or test a series of conditions one after the other.

## DO LOOP Statement

**Purpose:** The DOLOOP statement executes a block of statements while a condition is true.

**Syntax:**
```
DO WHILE (expression)
  statements...
LOOP
```

**Comments:** The *expression (condition)* logically evaluates to True or False.
The *statements* are the program lines between the DO and LOOP
The *statements* are repeated while the expression is True.
DO LOOPS may be nested.

**Example:**
```
x = 1
DO WHILE (Sqr(x) < 5)
  x = x + 1
MsgBox(x)
LOOP
```

Test this code in the DO_LOOP subroutine

1203-V6 – Introduction to Programming - v02 41 © LabWare 2011

- Sqr() is the square root function

# WHILE WEND Statement

**Purpose:** The WHILE WEND statement executes a series of statements in a loop as long as a given condition is true.

**Syntax:**
```
WHILE (expression)
  statements...
WEND
```

**Comments:** The *expression (condition)* must be in parenthesis.
The *statements* are repeated while the *expression* is True.

**Example:**
```
x = 1
y = 10
WHILE (x < y)
    MsgBox(x)
    x  = x + 1
WEND
```

Test this code in the WHILE_WEND subroutine

## FOR NEXT Statement

**Purpose:** The FOR NEXT statement executes a group of statements a specified number of times.

**Syntax:**
```
FOR i = 1 TO num STEP step
    statements...
NEXT
```

**Comments:** The *i* is a numeric variable to be used as a counter.
The *num* is the final value of the counter.
The *STEP* is the counter increment for each loop. If not specified, it is assumed to be 1.
The *statements* are the program lines between the FOR and NEXT

**Example:**
```
FOR i = 1 TO 10 STEP 2
    x = SQR(i)
    MsgBox(x)
NEXT
```

Test this code in the FOR_NEXT subroutine

1203-V6 – Introduction to Programming - v02 43 © LabWare 2011

- FOR NEXT loop
  - Statements are executed a defined number of times
- DO LOOP
  - Statements are executed until the condition is no longer valid
- WHILE WEND loop
  - Statements are executed until the condition is no longer valid
- **TIP: Make sure to step into ANY loop while testing using the LabWare Debugger**

## FOR NEXT Example

```
message = ""

FOR i = 1 to 10
    message = message + "This is line number " + i + Chr(10)
NEXT

MsgBox(message)
```

Test this code in the FOR_NEXT_2 subroutine



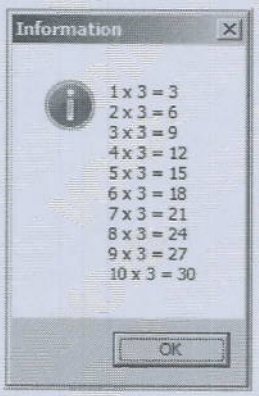
- FOR NEXT loops can be used to build multi-line messages.
- You will need to initialize your variable before the loop. In this case we do not want any extra spaces so we are just using double quotes with no space in-between.
- Every time the code runs through the loop "i" is increased by one (STEP is not defined therefore default STEP 1).

# Exercise LB01-10: FOR NEXT

In this exercise, you will write a subroutine that builds a "3" times table. Display the times table in a message box.

1 x 3 = 3
2 x 3 = 6
etc.

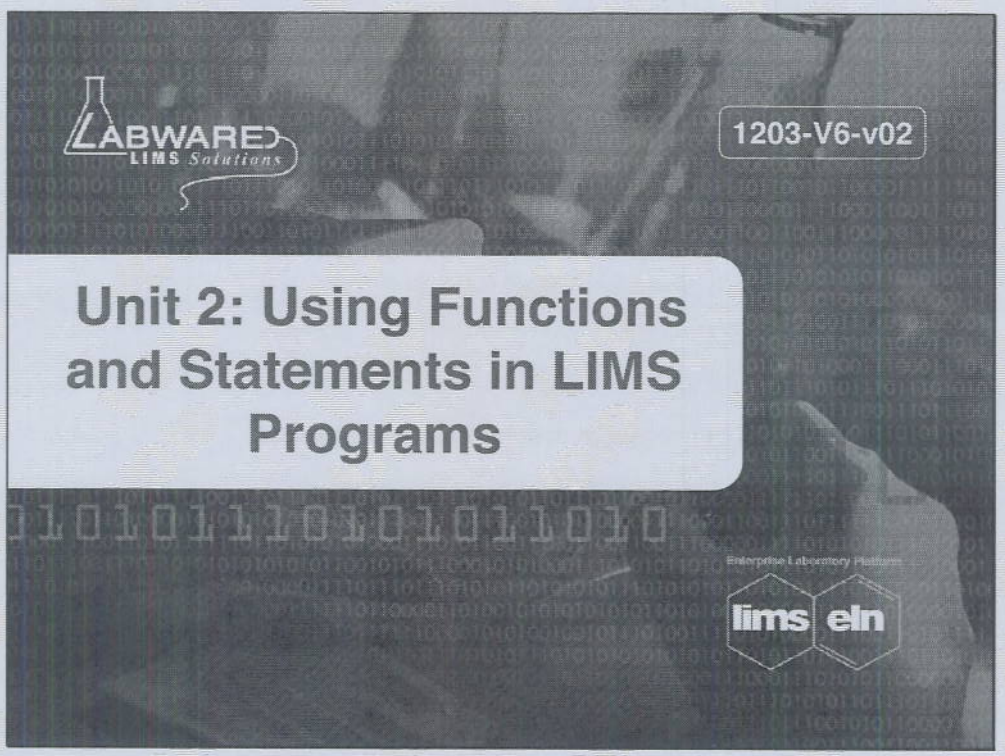
**Functions:**
**Chr()**
**MsgBox()**
**Statements:**
**FOR NEXT**




1) Create a subroutine named TIMES_TABLE

2) Use the code example given in the previous page to build the message box. The message box should look like this:

## Notes ...


LABWARE
LIMS Solutions
1203-V6-v02
Unit 2: Using Functions and Statements in LIMS Programs
Enterprise Laboratory Platform
lims eln

## Objectives

By the end of this segment, you should be able to:

1. Configure typical LIMS Basic functions
2. Execute SQL queries from a LIMS Basic subroutine
3. Understand arrays and use them in programming
4. Error trap your code to avoid unexpected errors
5. Comment and document your code




- This section introduces more LIMS Basic functions, including the SQL function.

New concepts
- LIMS functions
- SQL queries
- Arrays
- Error trapping
- Commenting and documenting code

New functions

AssignTest(), Mid(), CloseDialog(), CloseProject(), DateAdd(), ODBCDateTimeStamp(), Open Dialog(), DateTime(), PromptForYesNo(), Save(), SQL(), DateTimeAdd(), SQLSelect(), Str(), ExecuteQueryTag(), FileClose(), Ubound(), FileExists(), FileNew(), FileWrite(), UpdateDialog(), LogSample(), Wait(), UpdateTableFlds()

New statements
- GOSUB
- RETURN

# Section 1

# "LIMS" Functions

## LIMS Basic Functions

- LIMS Basic functions are organised in categories - e.g.
  - Arithmetic functions
  - String functions
  - Date functions
- Many of these functions are common to "BASIC" languages (Str, Left, Right, Std, Avg, Date, etc. ...)
- Others are specific to LabWare, for example functions to log samples or assign tests.

1203-V6 – Using Functions and Statements - v02 [4] © LabWare 2011

- We have already seen some functions that are common in BASIC languages.
- Some LIMS Basic function categories also contain:
  - Functions that are specific to LabWare objects such as a Batch, Folder, Lot, Sample, Test, and Result.
  - These functions perform LIMS operations such as updating a batch, logging a sample, and assigning tests.
  - Many of these commands involve making changes to the data in the database.

## "LIMS" Functions

Most LIMS-related functions have the following structure:

status = **function**(arg1,arg2,arg3,…,commitFlag)

| | |
|---|---|
| status: | a variable returning True (= success) or False (= failure) |
| **function**: | the function's name |
| arg(n): | the functions arguments (or inputs) |
| commitFlag: | a boolean flag (T or F) to indicate if the changes must be immediately committed (= saved) to the database |

1203-V6 – Using Functions and Statements - v02 [5] © LabWare 2011

**Status**

- Status is a variable that returns either **True** if the function was successfully executed or **False** if it failed.

**Arguments**

- LIMS-related functions often have numerous arguments; some optional and some mandatory.
- Define the arguments as variables BEFORE the function; not within the function itself.

**Commit Flag**

- The commit flag argument indicates whether the changes are immediately saved to the database or whether they are kept in memory to be saved later.

**Commit Flag = T**

- Changes are immediately saved to the database when the function is executed

**Commit Flag = F**

- Changes are kept in memory to be saved later, i.e. at the:
  - Next system commit
  - Next save() function
  - When the next function with the commitFlag = T is executed
- Keeping the changes in memory to be saved later is equivalent to a user clicking the cancel button at result entry.
- This allows the LIMS Basic to be configured with checks and error traps that will allow for scenarios where specific changes should not be written to the database.

## To Commit or Not Commit

- Some reasons why you may want to use commitFlag = "F":
  - Be able to opt out of committing the data
  - Commit all changes from a loop, in one operation after the loop
- The **Save()** function will commit all uncommitted data.
- Some older functions in the "LIMS" category do not have a commitFlag
  - Some of these functions will auto-commit (e.g. LogSample)
  - Others won't (e.g. AssignTest) therefore must be committed separately with the Save() function.

1203-V6 – Using Functions and Statements - v02 [6] © LabWare 2011

- Functions that do not have a commitFlag:
  - May either auto-commit or not commit
  - The only way to find out is to reference the LIMS Basic Manual
- Functions that are not committed (by choice or not) must be committed later in the code with the **Save()** function or one of the other ways listed on the previous page.

## Arrays

- An array is a variable that may contain more than one value, i.e. a data set
- Some LIMS Basic functions may have **array** type arguments.
- Two types of arrays:
  - One dimensional array (e.g. arrayName[1] = "a")
  - Multi dimensional array (e.g. arrayName[1,1] = "a")
- An array can consist of one or many elements.
- LIMS Basic code can be used to assigned values to the elements.

1203-V6 – Using Functions and Statements - v02 [7] © LabWare 2011

**Important Tip:**

- Refer to the LIMS Basic Manual to determine whether the array type argument is a one dimensional or multi dimensional array.

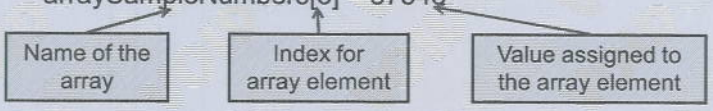
## One Dimensional Arrays

- Think of a one dimensional array as a list of values.
- Example of a one dimensional array:

  arraySampleNumbers[1] = 87543

  arraySampleNumbers[2] = 87544

  arraySampleNumbers[3] = 87545




- Elements of a single dimensional array are defined by a single index number, e.g [1].
- Elements of a multi dimensional array are defined by using one index number for each dimension, e.g. [1,1].




- The different values in the array are called array elements. In LIMS Basic an array can have as many elements as necessary.
- The array elements are defined using an index number within square brackets after the array name.
- Values can be assigned to the array elements

# One Dimensional Array: Sum()

- Function: x = Sum(array)
- Returns the sum of the values in the array
- Define the array elements and assign values

  array[1] = 10
  array[2] = 29
  array[3] = 39
  x = Sum(array)
  x = 78


- Think of the one dimensional array as a list of array elements
- Each element can hold a value

# Section 2

# Macro to log samples

## Function Demo: LogSample()

- sampleNumber = LogSample(templateName, fieldsArray, valuesArray)
- Logs a Sample and returns the new Sample Number (auto commit)

```
templateName = "CUST-PROD"

fieldsArray[1] = "CUSTOMER"
fieldsArray[2] = "PRODUCT"

valuesArray[1] = "ALPHA"
valuesArray[2] = "CHOC-CAKE"

sampleNumber = LogSample(templateName,fieldsArray,valuesArray)
```

- fieldsArray lists fields in the "CUST-PROD" Sample Login Template
- valuesArray lists the values to put in the fields when logging the sample

1203-V6 – Using Functions and Statements - v02 [11] © LabWare 2011

- The **sampleNumber = LogSample(templateName, fieldsArray, valuesArray)** function logs a Sample and returns the new Sample Number
- **sampleNumber** is the variable that will hold the new Sample Number
- **templateName** is the variable that holds the name of the Sample Login Template to be used to log the Sample
- **fieldsArray** is a one dimensional array that holds the Sample Login Template fields that will be populated with values
- **valuesArray** is a one dimensional array that holds the values that will be used to populate the corresponding fields on the Sample Login Template
- There is no limit on the number of array elements that can be defined in the fieldsArray and valuesArray; as long as values are valid for the selected template.
- The LogSample() functions includes an automatic commit

## Function Demo: AssignTest()

- This function can be used to assign a Test to a Sample

```
status =  AssignTest(sampNum, analysisName, numReps,
          "testNumberArrayName",fieldsArray, valuesArray, maxReps,
          parentTestNumber)
```

Example:

```
sampNum = 741
analysisName = "PH"
numReps = 3
fieldsArray[1] = "TEST_LOCATION"
fieldsArray[2] = "TEST_COMMENT"
valuesArray[1] = "BGM"
valuesArray[2] = "Test requested by customer."
status =  AssignTest(sampNum, analysisName, numReps, "testNumberArrayName",
          fieldsArray, valuesArray)
```

1203-V6 – Using Functions and Statements - v02 [12] © LabWare 2011

- The AssignTest function assigns a Test to a Sample.
- **sampNum** – the Sample Number of the Sample the Test will be assigned to
- **analysisName** – the name of the Analysis (Test) to be assigned to the Sample
- **numReps** – the number of Test replicates to be assigned
- **testNumberArrayName** – the name of the array containing the assigned Test Numbers
- **fieldsArray** – a one dimensional array containing the Test record field names to be updated by the function
- **valuesArray** – a one dimensional array with values with which the new Test records will be updated
- **maxReps** – the maximum number of Test replicates to be assigned to the sample
- **parentTestNumber** – Test number of the Test to act as a Parent Test, if null, the Test is assigned to the Sample

- In the slide example maxReps and parentTestNumber were removed from the function because they were not needed.

- Note also that there is no database commit as part of the AssignTest() function. You will need to use the Save() function to save the new tests to the database.

## Exercise LB02-01: Log a sample and assign tests

In this exercise, you will configure a subroutine to log a sample and auto-assign a test.

**Functions:**
**AssignTest()**
**LogSample()**
**Save()**
**Statements:**
**None**

1) Create a subroutine called MACRO_LOG_SAMP.

2) Use the LogSample() function to log a single sample using the CUST_PROD sample login template. Make sure that the following sample fields are populated with values:
   - CUSTOMER: ALPHA
   - PRODUCT: CHOC_CAKE

3) Use the AssignTest() function to auto-assign the MOISTURE analysis in duplicate to the sample. You will also update the following TEST fields:
   - TEST_COMMENT, to record that the test was added from LIMS Basic
   - INSTRUMENT, with an instrument name of your choice

4) Now test your code carefully.

## Error Trapping

- Most functions can be error trapped based on the value returned for the status variable.

  status = **function**(arg1,arg2,….)

- status = true: the function was executed successfully
- status = false: the function has failed

- lastError: system variable defined by LabWare when a LIMS Basic function failed

1203-V6 – Using Functions and Statements - v02 [14] © LabWare 2011

- Many of the LIMS Basic functions return a value of either true or false to the **status** variable.
- Furthermore, if the returned status is "False", LabWare provides a system variable called **lastError** that contains a high level indication / explanation of the error encountered.
- The status and lastError variable values can be used for error trapping.
- The value of the lastError variable:
  - Can be displayed to the user at the time that the error is encountered or it can be recorded in a log.
  - Will be displayed in the LIMS debugger window if you are using it to test your code.
  - Will be reflected in the debug.log file

## Exercise LB02-02: Error trapping

In this exercise, you will update the MACRO_LOG_SAMP subroutine to include an error trap on the test assignment.

**Additional functions:**
**MsgBox()**

**Statements:**
**IF THEN**

1) Check for the return status of the AssignTest() function.

   If the status is false, display a message to the user that indicates what the error is. You may use the lastError system variable. Your message should read:

   **No tests were assigned to sample ..... – Error: .....**

   If the tests have been successfully assigned, the message box should read:

   **Tests successfully assigned to sample .....**

2) Test your code.

1203-V6 – Using Functions and Statements - v02 [15] © LabWare 2011

## Exercise LB02-03: Log several samples in a loop

In this exercise, you will create a new subroutine to log several samples instead of just one. The code is similar to the previous exercise, but this time, you will start by prompting the user for a number of samples to log.

**Additional functions:**
**InputBox()**

**Additional Statements:**
**FOR NEXT**

1) Save your previous subroutine as MACRO_LOG_SAMPS
2) Prompt the user for a number of samples to log using an Input Box.
3) Use this information to set up a FOR NEXT loop that will log each sample and assign the MOISTURE test in duplicate.

## Exercise LB02-04: Build a multi-line message box

In this exercise, you will update the MACRO_LOG_SAMPS subroutine to display a message with information on ALL the samples logged. It should look like this:

```
Tests successfully assigned to sample 15
Tests successfully assigned to sample 16
Tests successfully assigned to sample 17
```

**Additional functions:**
**Chr()**

**Additional statements:**
**None**

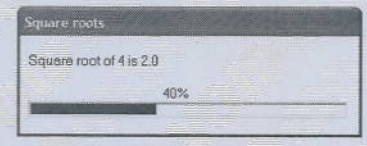
1) Remove all message boxes from the FOR NEXT loop.
2) Initialize the message BEFORE the loop (message = "")
3) For each logged sample, build one line of the message using the technique:

   message = message & "Tests successfully assigned to sample ….."

   Don't forget to add a carriage return at the end of each line. Of course, the message text will be different for tests that failed to be assigned.
4) Add a message box AFTER the loop to display your multi-line message.

## Progress Dialogs

- Progress Dialogs display the progress of an operation to the user and are often used in loops.




- 3 functions are required:
  - OpenProgressDialog
  - UpdateProgressDialog
  - CloseProgressDialog

You can test the above dialog in the PROGRESS_DIALOG subroutine

1203-V6 – Using Functions and Statements - v02 [18] © LabWare 2011

- Progress dialogs are vey useful for sections of code that are included in loops.
- Progress dialogs allow users to monitor the progress when the code is running thus users are less likely to believe that the system is "hanging".
- Progress dialogs consist of three functions
  - **OpenProgressDialog()** initializes the dialog
  - **UpdateProgressDialog()** is placed inside the loop and updates the progress
  - **CloseProgressDialog()** closes the dialog

## Function Demo: OpenProgressDialog()

**OpenProgressDialog()**

**Purpose**: The OpenProgressDialog function displays a progress dialog.

**Syntax**: status = OpenProgressDialog(title,message,showPercentage,allowCancelFlag)

**Comments**:
The allowCancelFlag is an optional argument

**Example**:

```
title = "Progress Dialog"
message = "Testing 1, 2, 3"
showPercentage = "T"
allowCancelFlag = "F"
status = OpenProgressDialog(title, message, showPercentage, allowCancelFlag)
```

**Returns**: status = true if the function was successful, false otherwise.

1203-V6 – Using Functions and Statements - v02 [19] © LabWare 2011

**OpenProgressDialog()**

- **title** - the displayed name in the title bar of the dialog window
- **message** – the displayed message in the dialog window
- **showPercentage** – if "T" (true) a % value and progress bar will be displayed in the dialog window
- **allowCancelFlag** – a boolean flag that indicates whether the user is allowed to cancel the current operation

## Function Demo: UpdateProgressDialog()

**UpdateProgressDialog()**

**Purpose:** The UpdateProgressDialog function updates the message and progress bar in a progress dialog.

**Syntax:** status = UpdateProgressDialog(message,percentage)

**Example:**

```
message = "Testing 4, 5, 6"
percentage = 20
status = UpdateProgressDialog(message, percentage)
```

**Returns:** status = true if the function was successful, false otherwise.




**UpdateProgressDialog()**

- **message** is displayed in the dialog box
- **percentage** displays a percentage value and controls the progress bar indicator display

## Function Demo: closeProgressDialog()

**CloseProgressDialog()**

**Purpose:** The CloseProgressDialog function closes a progress dialog.

**Syntax:** status = CloseProgressDialog()

**Returns:** status = true if the function was successful, false otherwise.

Have you


your progress dialog?

- It is very important to make sure that you close your progress dialog.
- The progress dialog will not close automatically leaving an open function.
- It is good practice to add all three functions to your code at the same time.

# Function Demo: Wait()

**Wait()**

**Purpose:** The Wait function causes the system to pause for a specified period of time

**Syntax:** status = Wait(seconds)

**Example:**

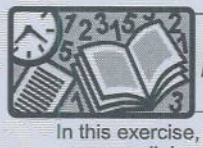
seconds = 3

status = Wait (seconds )

**Returns:** status = true if the function was successful, false otherwise

## Good Programming Reminders

- Comment your code
- Good comments make it easier to understand and debug code
- Use correct indentation
  - especially within loops



## Exercise LB02-05: Progress Dialog

In this exercise, you will update the MACRO_LOG_SAMPS subroutine, to include a progress dialog showing the progress of the sample logging.

**Additional Functions:**
**OpenProgressDialog(), UpdateProgressDialog(), CloseProgressDialog(), Wait()**
**Additional Statements:**
**None**

1) Insert a line just before the FOR NEXT statement to open and initialize the progress dialog.

2) The updating of the dialog should take place inside the FOR NEXT loop, towards the end. Ensure that the percentage calculation is correct. You may want to add the Wait() function.

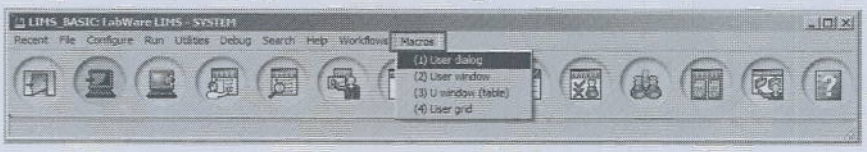
3) Don't forget to close the progress dialog at the end of the subroutine.

4) Add some comments. Make sure that the code inside the loop is correctly indented.

This subroutine has 3 distinct parts; prompting for a number of samples and initialize progress dialog, loop to log samples and assign tests, and close progress dialog and display message. Look at the exercise solution for tips on how to present your code.

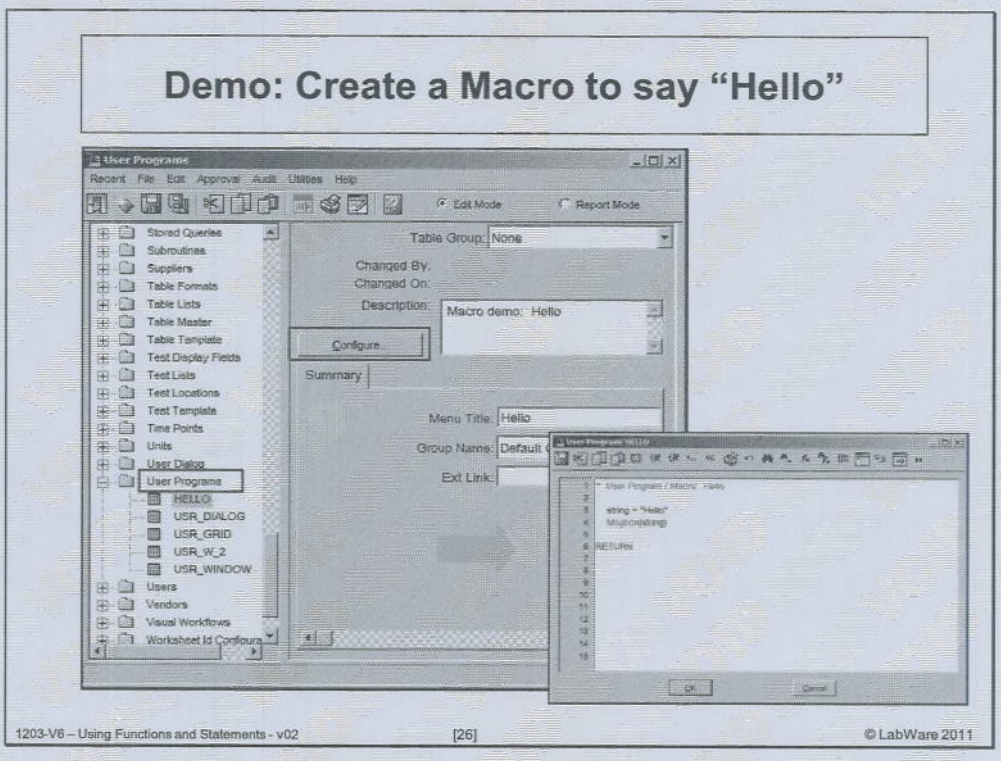
## Macros

- Macros are user-added menus that are available from the LabWare main menu.
- Macro records are stored in the User Programs table in LabWare.
- The LabWare toolbar needs to be configured for a user's role to make a Macro available
- Macros can call a Subroutine with the GoSub statement or the Subroutine function



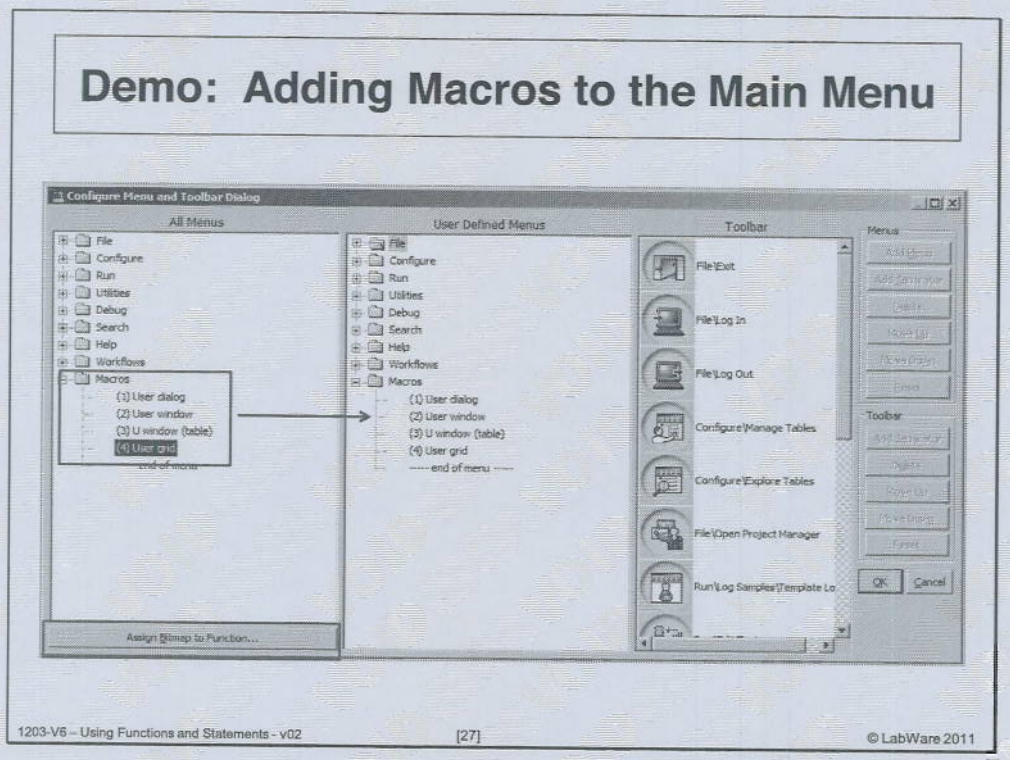

1203-V6 – Using Functions and Statements - v02 [25] © LabWare 2011

- Macros can be used to execute repetitive or frequently used tasks such as displaying reports or re-printing labels.
- Macros may also lessen the need to train users on LIMS functionality, for example if you create a sample login macro the user may not need to know how to log a sample, which template to select and whether to assign it to a project, lot or batch.
- **Important**:
  - Prior to visual workflows, macros were the only way to provide custom workflows.
  - In LabWare v6 it is recommended that visual workflows be used to provide custom workflows




## Creating a User Program

- Select the Explorer Table Manager icon on the LabWare Main Menu
- Select the USER PROGRAMS table
- Create User Program record with the name, "HELLO"
- Define the Menu title as "Hello" (This is the menu that will display in under the Macro Menu)
- Configure your MsgBox("Hello")




## Adding the Macros Menu Item to the LabWare Main Menu

- Log out of LabWare, log back in
- Select Configure > Menus and toolbar…
- Assign a bitmap to the macro (if a bitmap will be used)
- Drag the new menu under Macros in the All Menus pane into the User Defined Menus pane
- Drag bitmap onto Toolbar pane and click OK (if a bitmap will be used)
- If Macro is not displayed, log out and log back in

## GoSub Statement

**Purpose:** The Gosub statement is for execution of subroutines.

**Syntax:** **GoSub** SubName

**Comment:** Use the "Insert a Statement" icon to insert the GoSub statement 

**Example:** GoSub MYSUB

1203-V6 – Using Functions and Statements - v02 [28] © LabWare 2011

- Subroutines can be called from Macros using the GoSub statement.
  - Using subroutines instead of placing code directly within your user programs allow you to create generic subroutines that may be used in multiple places.
  - It also allows for more efficient change control. If your subroutine needs updating you only need to update one area instead of each individual user program.
- We will discuss another function to call a subroutine, Subroutine(), in the next unit.

>>If time, demo how a visual workflow would be used for the same subroutine.

## Exercise LB02-06: Set up a macro

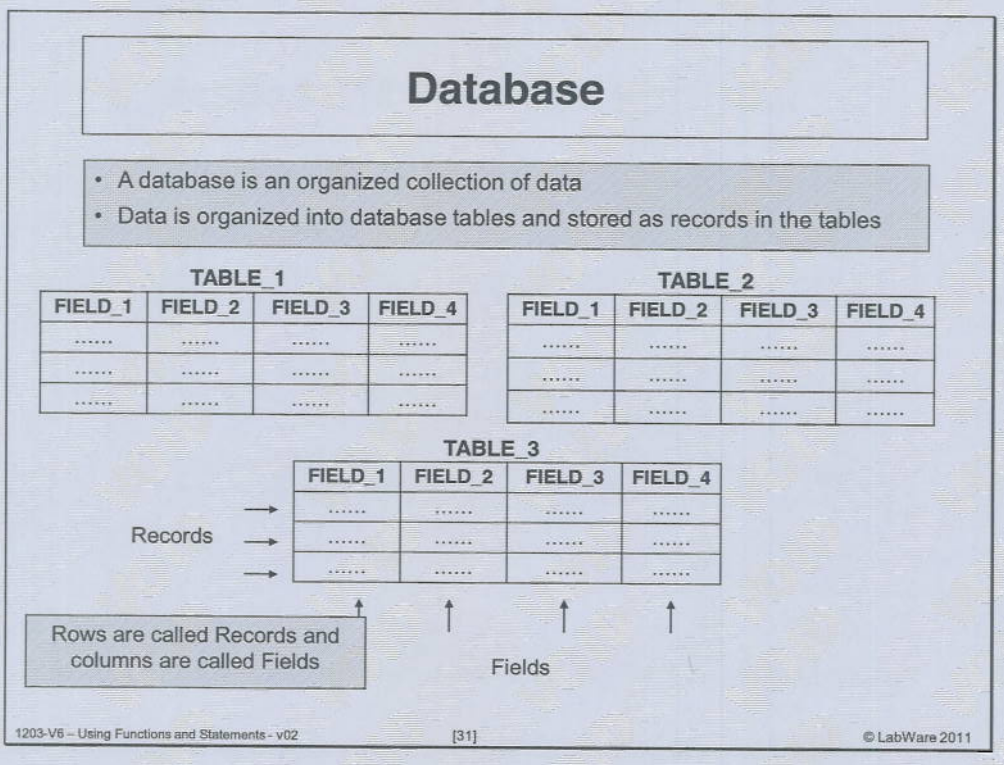
You can now configure a macro to execute the code that you have configured in the previous exercise.

**Functions:**
**None**
**Statements:**
**GOSUB**

1) Create a new User Program called LOG_SAMPS. The menu title for the macro should be displayed as "Log samples".
2) Insert the GOSUB statement to call the MACRO_LOG_SAMPS subroutine and save.
3) Log out of LIMS and back in.
4) Configure the LabWare Main Menu to display the Macros Menu Item.
5) Test the Macro.

# Section 3

# Running SQL from LIMS

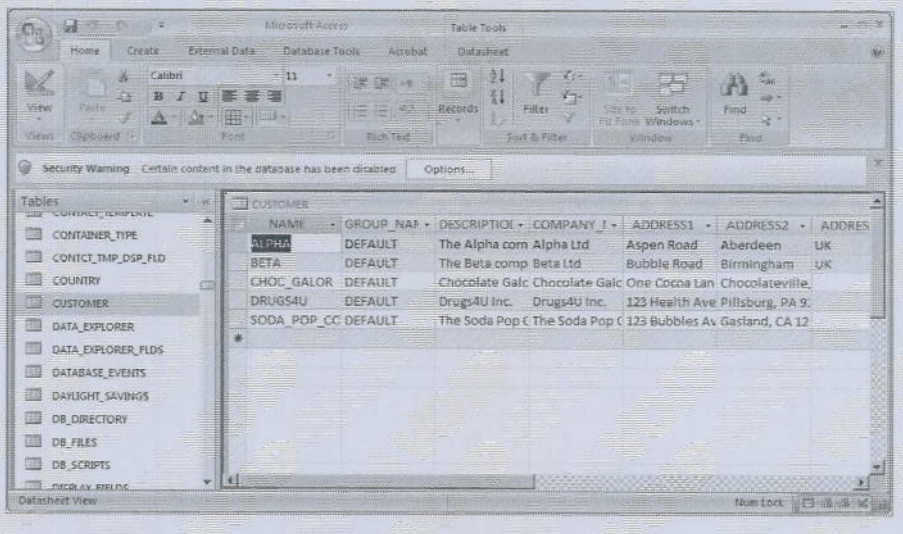



- There are many instances in LIMS Basic when you will need to query the database and process the result of that query.
  - For instance, you may want to list all the grades for a product in a dropdown list, or you may want your code to find out which samples belong to a certain location so that you can add them to a project.

- The following few slides provide a short review of SQL principals that were discussed in LIMS Admin 1 training and that you may have used to build LabWare query tags.

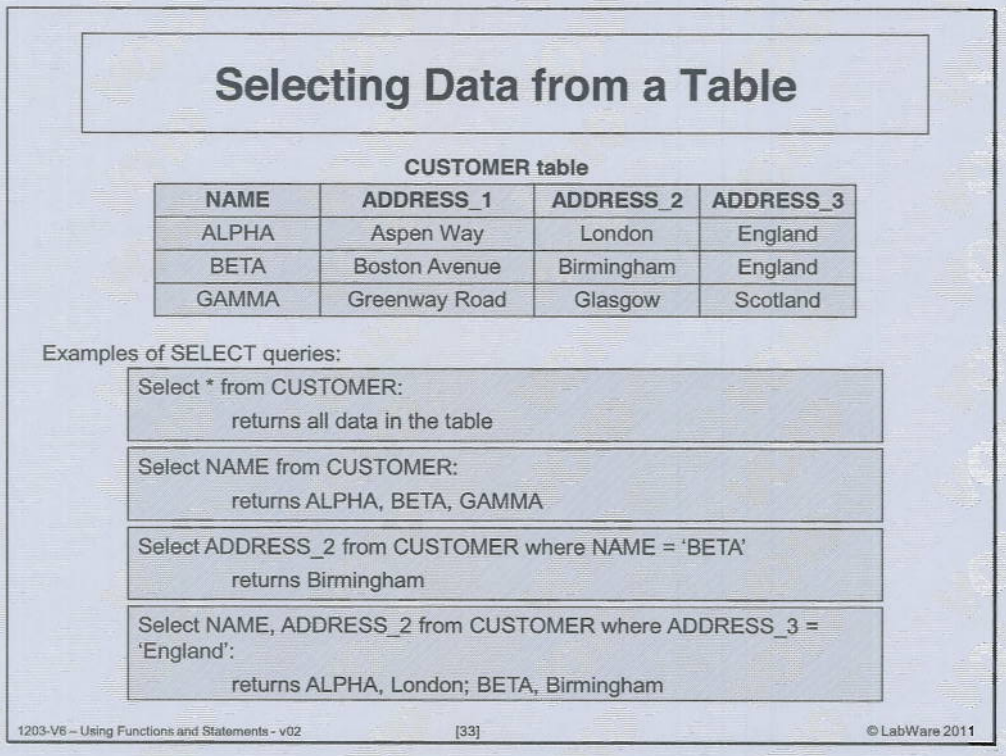
Reminder: SQL stands for Structured Query Language. It is a generic language used to query or modify databases.

## LabWare Tables in MS Access




1203-V6 – Using Functions and Statements - v02 [32] © LabWare 2011

- The above table is an example of a database table.
- The name of the table is CUSTOMER
- Data regarding customers is stored as individual records in the table, e.g. ALPHA, BETA, etc.




- This slide is an example of how a SQL query works. The selected data (or query result) will be returned in a multi-dimensional array.
- Using **Select * from (TABLE)** returns all data from that table. This is useful when you are not sure what kind of data is stored in each of the table's fields. For example, to determine how the ADDRESS_3 field is being used.
- When selecting a FIELD from a TABLE, the return will be an array.
- If you specify a where clause, the text value must be in single quotes.

## SELECT Query

SELECT NAME, ADDRESS_2 -The selected fields

FROM CUSTOMER -The table

WHERE ADDRESS_3 = 'England' -The criteria that records have to match to be selected

Text values in the WHERE clause must be framed with single quotes

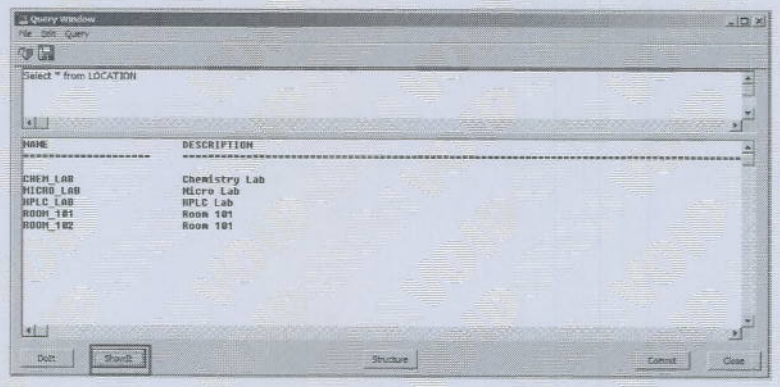
1203-V6 – Using Functions and Statements - v02 [34] © LabWare 2011

- Select SQL queries generally follow the syntax:

Select FIELD_1, FIELD_2, FIELD_3….
From TABLE_NAME
Where FIELD_4 = *numeric value* and FIELD_5 = '*text value*'

## Demo: Query Window

- SQL queries can be executed from the Query Window
- Select **Debug > Query Window** to open the window.
- Type query in the top pane.
- The selected data is returned in the bottom pane by clicking on the **ShowIt** button




1203-V6 – Using Functions and Statements - v02 [35] © LabWare 2011

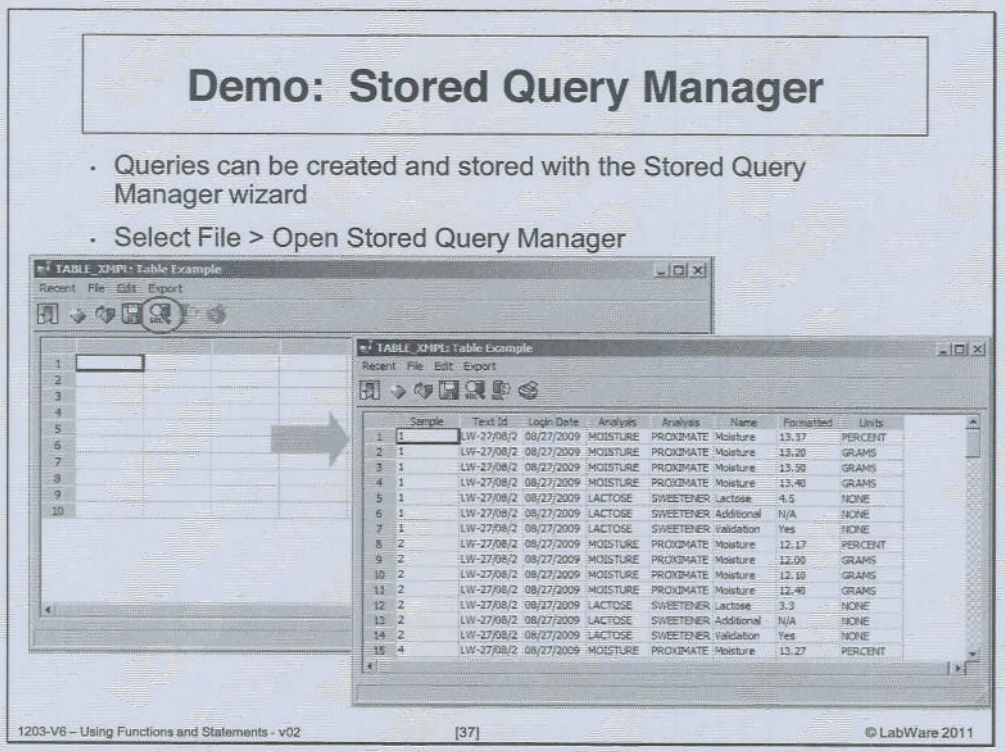
- The Query Window is a useful tool when developing LIMS Basic.
- The **Structure** button will display tables and fields. In the next few slides we will look at other areas queries can be written.
- Important: Only execute SELECT type SQL Queries from the LabWare Query Window

## Exercise LB02-07: The Query Window

In this exercise, you will create and run a SQL query from the Query Window.

1) Open the Query Window and configure a query that will:
   select the
   SAMPLE_NUMBER
   TEXT_ID
   PRODUCT
   from the SAMPLE table for samples
   where the PRODUCT value was defined as CHOC_CAKE

2) Click on the ShowIt button to display the data.




## Using Stored Query Manager

1) File > Open Stored Query Manager
2) Select New Query; Enter name for query
3) Highlight the table you are selecting from. Click Next
4) Click Add button to define fields you want to report on. Click Next
5) Click Add button to define fields for the "where" clause. Click Next
6) Define the values for the "where" clause. Click Next
7) Define a sort order (optional)
8) Limit number of returned records (optional)
9) The resulting query is displayed. Click Finish

- To execute a Stored Query click on Perform Query or select File > Run Query. Once the query is executed, the results can be exported to Excel.

## Exercise LB02-08: The Stored Query Manager

In this exercise, you will create and run a SQL query from the Stored Query Manager.

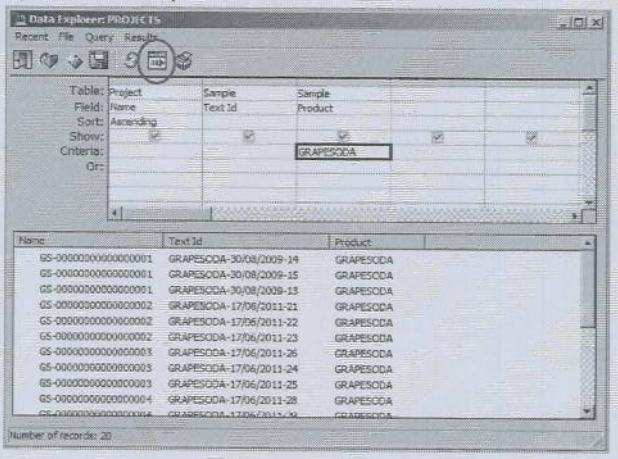
1) Open the Stored Query Manager and build the same query as in the previous exercise – i.e.
   select the
   SAMPLE_NUMBER
   TEXT_ID
   PRODUCT
   from the SAMPLE table for samples
   where the PRODUCT value was defined as CHOC_CAKE

2) Save the query.

3) Click on the "Perform Query" button to display the data.

## Demo: Data Explorer

- The Data Explorer is another place in LIMS where queries can be created, stored and run from.
- File > Open Data Explorer




1203-V6 – Using Functions and Statements - v02 [39] © LabWare 2011

## Using Data Explorer

- File > Open Data Explorer
- Select the fields first by table, then select the field from the drop down in each column/row
- Enter your "where" clause
- Click on the Run icon to run the query
- To view the SQL select Query > Edit SQL

- The data explorer allows users to export the results either to Excel or to an adhoc folder. Data Explorer allows the use of prompts for your "where" clause.

- Data Explorer and Store Query Manager are useful tools to assist with trending and quick information gathering that end users may be granted restricted access to.

## Exercise LB02-09: The Data Explorer

In this exercise, you will create and run a SQL query from the Data Explorer.

1) Build the same query as in the previous exercise in the Data Explorer.
2) Save the query.
3) Click on the "Run Query" button to display the data.