



LotusScript basics— the foundation

- 2.1 Chapter objectives 12
- 2.2 Variables 13
- 2.3 Identifiers 18
- 2.4 Looping 19
- 2.5 Other methods to affect code execution 22
- 2.6 Interacting with the user 25
- 2.7 Commenting your code 29
- 2.8 Strings 29
- 2.9 Format function 33
- 2.10 Predefined constants 34
- 2.11 UnderScore (continuation) 34
- 2.12 Chapter summary 35
- 2.13 Chapter review 35

2.1 *Chapter objectives*

- Introduce syntax of the LotusScript language
- Variable types
- Constants
- Options section
- Declarations section
- Converting data from one type to another
- Public/Private/Static identifiers
- `For` loops
- `Do` loops
- `While` loops
- `Forall`
- Branching (If/Then and Select Case)
- Displaying messages
- User input
- Commenting your code
- Strings
- The Format function
- Predefined constants
- Continuation

Learning a Basic-like language is much easier than grasping arcane languages like C or C++. Basic has English-like statements. It was designed with ease of use in mind. This has led to the enormous popularity of languages like QBasic and Visual Basic from Microsoft Corporation.

The syntax of LotusScript is almost identical to Visual Basic which makes it easy for Visual Basic developers to move to LotusScript although they still have to learn the environment. Working with Lotus Notes/Domino requires a thorough understanding of such elements as views, databases, forms, and agents.

This chapter gives an overview of the basics of developing with LotusScript and Lotus Notes/Domino. If you are familiar with Visual Basic, you may want to skim the material. You may be able to skip this section if you are already familiar with the environment, but that decision is up to you.

2.2 Variables

LotusScript has seven data types: variant, integer, double, long, single, string, and currency. Integer and long hold numbers with no decimal places. Single and double store decimal numbers (real numbers); string variables store text and currency holds monetary values.

2.2.1 Data Type Minimum/Maximum Value

LotusScript defines the following range of values for each numeric data type:

Table 2.1

Date Type	Minimum Value	Maximum Value
Integer	-32,768	32,767
Long	-2,147,483,648	2,147,483,647
Single	-3.402823E+38	3.402823E+38
Double	-1.7976931348623158E+308	1.7976931348623158E+308
Currency	-922,337,203,685,477.5666	922,337,203,685,477.5666

The string type is limited as well. The size of a string value is limited to 64K, and a string literal is limited to 32K.

2.2.2 Declaring and using variables

The Dim keyword (dimension) is used to declare a variable. Once the variable has been declared, it can be used.

```
Dim variable_name As variable_type
```

Variables can be declared anywhere within a script. The As keyword and variable_type are not required when declaring variables of the type variant. Variant, a special type of variable, will be addressed in an upcoming section.

To declare an integer called index:

```
Dim index As Integer
```

To declare a string called name, and set its value:

```
Dim Name As String  
Name = "Tony Patton"
```

Multiple variables of the same type can be declared on one line (they are separated by commas), but the type of each must be included.

Note E represents scientific notation, it signals the number of leading (-) or trailing (+) zeroes.)

```
Dim index1 As Integer, index2 As Integer, index3 As Integer
Dim Name1 As String, Name2 As String
```

Note If multiple variables are declared on one line and any of the variables is missing a type declaration, it is declared as a `Variant` data type. `Variant` is the default data type. For instance, the following line

```
Dim var1, var2 As String
```

declares `var1` as a `Variant` and `var2` as a `String` variable.

2.2.3 Constants

Constants are variables whose value does not change; it is constant. The keyword `CONST` is used to define a constant in a script.

Note LotusScript constants are usually defined in all capitals.

```
Const CONSTANT_NAME = constant_value
```

There are a number of built-in constants in LotusScript:

Table 2.2

EMP	The initial value for a variable of the type <code>Variant</code> .
FALSE	Returned by comparison or logical operations; it has the numeric value of 0.
NOTHING	Objects are initialized to <code>Nothing</code> . This is used extensively in loops or condition statements when working with <code>Notes</code> objects.
NULL	A <code>variant</code> variable that has no data. It can be used to retrieve memory occupied by a <code>variant</code> by setting the <code>Variant</code> to this.
PI	3.14159265358979...
TRUE	Returned by comparison or logical operations; it has the numeric value of -1.

2.2.4 Explicit/Implicit

By default, LotusScript allows a variable to be referenced/used before it has been declared. If this happens, the variable is declared as a `variant`. This can lead to messy code. This type of *freedom* is called implicit declarations. The developer is allowed to implicitly use a variable whenever he or she likes.

Explicit declaration is the opposite of implicit. It demands that a variable be declared before it is ever used. That is, it must have a `Dim` statement that declares it. This leads to much more readable and manageable code. The `Option Explicit` line can be entered into the Options section of a script to enforce explicit declarations. `Implicit` declarations is the default.

```
[Options]
Option Explicit
```

If a script is set up to use explicit declarations, a compile-time error is generated if a variable is referenced before it is declared.

Example 2.1

```
[Options]
Option Explicit

[Initialize]
Sub Initialize
    For x = 1 To 10
        MsgBox x
    Next
End Sub
```

Explanation

Attempting to save this script generates a compile-time error. The Errors box contains the following statement:

```
ExplicitTest: Initialize: 2: Variable not declared: X
```

- The name of the script is `ExplicitTest`
- The error occurred in the `Initialize` event.
- The error number is 2.
- The error is `Variable not declared: X`.

The variable `x` must be declared before it is used as the index in the `FOR` loop.

2.2.5 Options

In addition to setting implicit/explicit variable declarations, the options section of a script allows other items to be set up. The default lower bound for an array is zero (just like languages such as C or Java), but the `Option Base` statement is used to reset the lower bound.

```
Option Base X
```

This statement sets the default lower bound for arrays to the number `X`. Chapter 5 covers arrays in more detail. `Option Compare` is used to configure the way string comparisons are handled. The following list shows its various formats.

- `Option Compare Binary` says that string comparisons will be performed at the bit level. A case-sensitive, pitch-sensitive comparison will be performed. This is used alone, no other `Option Compare` statements can exist.

- `Option Compare Case/NoCase` allows you to specify if string comparisons will be case-sensitive. This can be used with other `Option Compare` statements, but only one `Option Compare Case/NoCase` can exist.
- `Option Compare Pitch/NoPitch` specifies if pitch-sensitive comparisons are used. This can be used with other `Option Compare` statements, but only one `Pitch/NoPitch` can exist.

Option Public

If `Explicit` variable declarations are used, the `Option Public` statement tells the system that each variable declared is `Public` unless otherwise specified. This overrides the system default of `Private`.

Option Declare

Another way to signal `Explicit` variable declarations is the use of `Option Declare`. This tells the system that implicit declarations are not allowed. It performs the same function as `Option Explicit`.

2.2.6 *Converting from one type to another*

LotusScript includes a number of functions to handle the conversion of one data type to another.

- `CCur(number or string)` Converts numeric value rounded to a four decimal place currency value
- `CDat(number or string)` Converts value to a date/time value
- `CDBl(number or string)` Converts numeric value to a double value
- `CInt(number or string)` Converts numeric value to a rounded integer value
- `CLng(number or string)` Converts numeric value to long value
- `CSng(number or string)` Converts numeric value to single value
- `CStr(number or string)` Converts value to a string value
- `CVar(any value)` Converts value to a variant value

Example 2.2

```

Dim var1, var2 As Integer
Dim var3 As Double
Dim var4 As Currency
Dim var5 As String
var2 = 250
var3 = 20.00
1 var4 = CCur(var2)
2 var1 = CInt(var3)

```

```
3 var5 = CStr(var2)
```

Explanation

- 1 The value of `var2` is converted to a currency value and stored in `var4`.
- 2 The value of `var3` is converted to an integer value and stored in `var1`.
- 3 The value of `var2` is converted to a string value and stored in `var5`.

2.2.7 The special variable type: Variant

If you are ever unsure of what type of data a variable may need to store, declare it as a `Variant`. `Variant`s can store any type of variable but a user-defined type. It can store all types of numbers, string values, arrays, dates, lists, or objects. `Variant` is the default data type in the Notes environment. If a variable is defined with no type (`Dim` variable) or implicitly used, it is set to a `Variant`. To declare a `Variant`:

```
Dim var  
- OR -  
Dim var As Variant
```

If a variable is implicitly used, it is set to a `Variant`.

Note `Variant`s require more resources than other data types. This includes memory, disk space, and processing time. Always specify a data type for variables whenever possible.

2.2.8 Determining the type of a variable

You may find yourself working with a `Variant` variable and need to know immediately what type of data it contains. This could occur for a number of reasons, but it is easily overcome. Two statements in LotusScript identify the type of value(s) stored in a `Variant` object: `DataType` and `TypeName`. Both accept the variable name and return the variable type. The difference lies in the format of the returned value. `DataType` returns an integer that corresponds to the data type, and `TypeName` returns a string identifying the type.

Example 2.3

```
1 Dim index  
2 index = 2  
3 Print DataType(index)  
4 Print TypeName(index)
```

Details:

- 1 A `Variant` called `index` is declared
- 2 The `Variant` is set to the integer 2.

- 3 The `DataType` statement is used to determine `Variant`'s data type. It prints 2 for integer.
- 4 The `TypeName` statement is used to determine `Variant`'s data type. It prints `INTEGER`.

The LotusScript online Help contains more details concerning the return values for both functions.

2.3 Identifiers

Identifiers precede variable declarations to signal the visibility of the variable. The two types of visibility for variables in LotusScript are `Public` and `Private`. `Private` is the default identifier, and it is used if no identifier is specified. `Private` restricts access to a variable to the section of code in which it is defined. If a private variable is in a function or subroutine, it is only visible within that function or subroutine. On the other hand, `Public` opens a variable to everything.

The `Static` keyword signals that a function/subroutine's local variables will retain their value(s) between calls to it. Normally, a variable defined in a function/subroutine will exist in memory only when the function/subroutine is accessed in the script.

Note Functions and subroutines are covered in chapter 4.

Example 2.4

```
1 Public varPublic
2 Private varPrivate
```

Explanation

- 1 The variable `varPublic` is declared as visible to everything in the script in which it is used.
- 2 The variable `varPrivate` is declared as visible to only those objects in its section of code.

Example 2.5

```
1 Static Public Sub TrackHits
2 Private count As Integer
3   count = count + 1
   End Sub
```

Explanation

This function tracks the number of times an item is accessed. We don't want the counter reset each time the code is accessed, so the `Static` keyword is used to make sure the variables will retain their values between calls.

- 1 The subroutine TrackHits is declared so that it can be accessed by all (`Public`) and the values of its local variables will be retained between calls to it (`Static`).
- 2 The local variable count is declared as an Integer.
- 3 The count variable is incremented each time the subroutine is accessed.

2.4 Looping

Four types of loops can be used in LotusScript: `For`, `Do`, `While`, and `Forall`.

2.4.1 For

The `For` loop allows you to execute a block of code a certain number of times.

Syntax

```
For [count variable] = [start value] to [end value] step [increment value]
  <block of code>
Next [count variable]
```

The count variable, start value, end value, and increment value are all numeric values. The step and its corresponding increment value are optional, as is the count variable on the next statement. Example 2.6 prints integers 1 through 5.

Example 2.6

```
For x = 1 to 5
  Print "x = " & x
Next x
```

Output

```
x=1
x=2
x=3
x=4
x=5
```

You will notice that implicit declarations are used, since the variable `x` is used before it is declared. Example 2.7 prints the multiples of two up to twenty.

Example 2.7

```
For x = 0 to 20 step 2
  Print "x = " & x
Next x
```

Output

```
x=0
x=2
```

```
x=4
x=6
x=8
x=10
x=12
x=14
x=16
x=18
x=20
```

The loop counts from 0 to 20, but it does so in increments of 2 (steps). `FOR` loops can be nested to achieve numerous results.

Example 2.8

```
For x = 1 to 2
  For y = 1 to 5
    Print "x = " & x " and y = " & y
  Next y, x
```

Output

```
x= 1 and y= 1
x= 1 and y= 2
x= 1 and y= 3
x= 1 and y= 4
x= 1 and y= 5
x= 2 and y= 1
x= 2 and y= 2
x= 2 and y= 3
x= 2 and y= 4
x= 2 and y= 5
```

2.4.2 Do

The `DO` loop allows the execution of a block of code until a condition is met. Its main strength is in allowing the execution of code before a condition is checked. The `DO` statement is used in conjunction with the `While`, `Until`, and `Loop` statements. It has four variations:

- 1 `Do While [condition]`
 <block of code>
 `Loop`
- 2 `Do Until [condition]`
 <block of code>

```

Loop
3 Do
    <block of code>
Loop Until [condition]

4 Do
    <block of code>
Loop While [condition]

```

Example 2.9

```

Dim response As String
response = "Continue"
Do While (response <> "Done")
    response = InputBox$("Please enter response")
Loop

```

This loop will continue until the word Done is entered by the user. It could have been coded a different way.

Example 2.10

```

Do
    response = InputBox$("Please enter response.")
Until (response = "Done")

```

Example 2.10 accomplishes the same task, but with two fewer lines of code. The beauty of the `Do` loop lies in Example 2.10; it allows the execution of a block of code before a condition is checked. We had to initialize the variable in 2.9, but 2.10 allows us to let the user set the variable before it is checked.

2.4.3 While

The `while` loop executes a block of code until a condition is validated as `true`. The block of code in the loop begins with the `while` statement and ends with the `wend` statement.

```

While [condition]
    <code>
Wend

```

Examples 2.9 and 2.10 could have been coded using a `while` loop; it strongly resembles example 2.9.

Example 2.11

```

Dim response As String
response = ""
While (response <> "Done")
    response = InputBox$("Please enter response.")
Wend

```

2.4.4 Forall

The `Forall` statement executes a block of code repeatedly for each item in an array or list. It has the following syntax:

```
Forall <variable> In <array/list name>
  <do something>
End Forall
```

Chapter 5 uses the `Forall` statement extensively.

2.5 Other methods to affect code execution

2.5.1 If

The `If` statement checks a condition first, and executes a block of code *if* the condition evaluates to `true`. It can be used to check a number of conditions with additional conditions contained in `ElseIf` statements, and a default action(s) contained in an `else` block. It has the following syntax:

```
If [condition] Then
  <code>
ElseIf [condition]
  <code>
Else
  <default action>
End If
```

There can be any number of `ElseIf` statements, but only one `If` and `Else`. The `else` condition is not required; there does not have to be a default action. The `If` loop differs in the fact that it is executed only one.

Example 2.12

```
Dim response As String
response = InputBox$("Please enter response.")
If (response = "1") Then
  Print "Reponse was 1."
ElseIf (response = "2") Then
  Print "Response was 2."
ElseIf (response = "3") Then
  Print "Response was 3."
Else
  Print "Response was neither 1, 2, or 3."
End If
```

Explanation

Example 2.12 accepts input from the user, and checks the input for certain values and performs certain actions depending on the value entered (or not entered).

2.5.2 *Select Case*

The `select case` statement mimics the behavior of the `if` statement. `Select case` will execute a block of code depending on the value of an expression. It has the following syntax:

```
Select Case variable
  Case condition :
    <statements to execute>
  Case Else
    <statements to execute>
End Select
```

Here is the select case representation of example 2.12.

Example 2.13

```
Dim response As String
response = InputBox$("Please enter response.")
Select Case response
  Case "1" : Print "Response was 1."
  Case "2" : Print "Response was 2."
  Case "3" : Print "Response was 3."
  Case Else : Print "The response was neither 1, 2, or 3."
End Select
```

2.5.3 *Stopping execution*

The `Exit` and `End` statements are used to stop execution, but `Exit` stops execution of the current block of code and `end` stops the current script altogether. There are many variations of the `Exit` statement:

- `Exit Do`
- `Exit For`
- `Exit Forall`
- `Exit Function`
- `Exit Property`
- `Exit Sub`

Each stops execution of the loop that follows it. Of course, they are only valid if they are contained within the type of loop specified.

Example 2.14

```
For x = 1 to 10
  value = InputBox$("Enter a value.")
  If (value = "") Then
1    Exit For
```

```
End If
Next x
```

Explanation

- 1 The script exits the `For` loop if a blank value is entered by the user.

2.5.4 Goto

No command has received more negative comments than the `Goto` statement. This statement allows the execution of a script to be transferred to another point in the script. Often it is linked to spaghetti code, but this is due to its use in procedural programming. The `Goto` can be a great aid in development if appropriately used.

Syntax

```
Goto <Label>
```

The label designates a point in a script. A label resides on its own line with a colon appended. It looks like this:

```
Handle_goto:
MsgBox "This was accessed with a goto statement."
```

`Handle_goto:` is a label. Once a label is defined, control can be transferred to that point in the script from anywhere else using a `Goto` statement. Many developers view the use of the `Goto` statement as bad programming, because it can make a program hard to read and debug. Of course, this is true if the `Goto` statement is used to jump all around a script. On the other hand, LotusScript requires that it be used when handling errors (see chapter 7).

Example 2.15

```
response = InputBox$("Please enter a name.")
If (response = "") Then
1   Goto No_Response
End If
...
2   Exit Sub
3   No_Response:
    MsgBox "You entered a blank name.  Aborting..."
4   End
```

Explanation

- 1 If the response variable is empty, control is transferred to the section of code with the `No_Response` label.
- 2 The `Exit Sub` statements are placed before the `No_Response` label, so that its contents are executed only when it is called.

- 3 The No_Response label.
- 4 The script ends.

2.6 Interacting with the user

Numerous situations arise where user input is needed, or information must be displayed to the user. LotusScript provides the `Print`, `MessageBox`, and `InputBox` statements to handle these situations.

2.6.1 Print

LotusScript defines the `Print` statement as sending output to the screen. In the Notes/Domino environment, the `Print` statement sends its output to the status line at the bottom of the Notes Workspace. If it is used in scheduled agents, the output is sent to the Notes Log. Finally, when used with web-related applications, it sends the output to the web browser (so it acts like the definition in LotusScript Help). Chapter 18 gives more details about interacting with web browser clients. The format is very simple.

```
Print <expression>
```

Expression can be a string, number, or date/time value. `Print` is great for keeping users informed when a script is running in the foreground.

2.6.2 MessageBox

`MessageBox` has two variations: one returns a value (a function) and the other does not (statement). The format of the two is almost identical, the function has the following syntax:

```
return_value = MessageBox ( message_to_the_user, (buttons + icon + default + mode), messagebox_title)
```

The format without a return value has the same syntax minus the parentheses:

```
MessageBox message_to_the_user, (buttons + icon + default + mode),  
messagebox_title
```

The only item required in each variation is the the message to be displayed. A title for the box is not necessary, and the default OK button will be displayed with no icon if nothing is specified.

Note The `MessageBox` keyword can be abbreviated to `MsgBox` with no errors.

Note Constant values can be used for the buttons, icon, default, and mode parameters of the `MessageBox` statement/function. This is detailed later in the chapter.



Figure 2.1
MessageBox
with Hello
message

Example 2.16

MessageBox "Hello"

The second parameter determines the buttons that appear on the MessageBox, the icon that is displayed, the default button (if the user clicks Return), and the modality. Various integer values correspond to different button combinations, icons, default buttons, and modality. The numbers for each setting are added together to set the second parameter of the MessageBox statement/function. The numbers and their settings follow.

Table 2.3

Buttons	
Button Type	Integer Value
OK	0
OK and Cancel	1
Abort, Retry, and Ignore	2
Yes, No, and Cancel	3
Yes and No	4
Retry and Cancel	5
Icons	
Icon Name	Integer Value
Stop Sign	16
Question Mark	32
Exclamation Mark	48
Information	64
Default Button	
Button Designation	Integer Value
First Button	0
Second Button	256
Third Button	512
Modality	
Modal Type	Integer Value
Application Modal	0
System Modal	4096

So, make a selection for each category (buttons, icon, default button, and modality) and add each integer value together for a final integer value that is used in the MessageBox statement/function.

Example 2.17



Figure 2.2 MessageBox

Let's say I want to display a warning message to the user with OK and Cancel buttons (figure 2.2). The default button will be Cancel, and the modality will be application modal. This means the the current application will stop running until the user responds, system modality stops all applications on the system until the user responds. Finally, a message will be displayed with the title of Error.

displayed with the title of Error.

```
MessageBox "There has been an error", (1 + 16 + 256 + 0), "Error"
```

If I want to know which button a user selects, I would use the MessageBox function as shown in figure 2.3.



Figure 2.3 User selected OK button in MessageBox

```
value = MessageBox("There has been an error.",273,"Error")
If (value = 1) Then
    MsgBox "The user hit the OK button"
ElseIf (value = 2) Then
    MsgBox "The user hit the CANCEL button"
End If
```

A return value of one signals the OK button was selected, and two signals the Cancel button. A list of return codes is in table 2.4.

Table 2.4

Button	OK
Cancel	Abort
Retry	Ignore
Yes	No
Return Value	1
	2
	3
	4
	5
	6
	7



Figure 2.4
Multiple line
messages

Multiple lines can be displayed as well (figure 2.4). The `Chr` function can be used to insert carriage returns in the message displayed in the message box.

Example 2.18

```
MessageBox "This is line one." & Chr(10) & "This is line two." _
& Chr(10) & "This is line three." & Chr(10) & "This is line" _
& "four.", 0, "Multiple lines"
```

`MessageBox` is great for presenting information to the user, but retrieving data is another function.

2.6.3 InputBox

There is only one format for the `InputBox` function, but it accepts from one to five parameters. The format is like this.

```
value = InputBox(message_to_user, title, default_value, display_column,
display_row)
```

The only item required is the message to the user; it can be used to tell the user what type of information to enter.

Example 2.19



Figure 2.5 InputBox

```
something = InputBox("Please " _&
"enter something.")
```

The box from example 2.19 can be “cleaned up” by adding a title and a default value, and telling the system where to display it on the screen (figure 2.5).

Example 2.20



Figure 2.6 InputBox

```
something = InputBox("Please " _&
"enter something.", "Enter Value", _
"LotusScript", 30, 30)
```

The last two parameters signal the *x* and *y* coordinates for the upper left corner of the `InputBox`. Example 2.21 tells the system to display an input box (figure 2.6) with its upper left corner starting at the coordinate (30,30).

Note The return value of the `InputBox` function is a variant, but a dollar sign can be appended to the `InputBox` statement to signal a string value:

Example 2.21

```
something = InputBox$("Please enter a string value.")
```

This accepts input from the user in the form of a string.

2.7 Commenting your code

Good programming practice involves the insertion of comments that describe the purpose of code and give specific details on items that may be tricky. Good comments can serve the purpose of refreshing your memory six months from now, or providing a roadmap for a new developer on the job once you have moved on to a new project. Be aware that the overuse of comments can be as bad as no comments at all. Comments are inserted in LotusScript using two methods: The %REM..%END REM block or using an apostrophe.

Example 2.22

```
%REM
  Author:  Tony Patton
  Date:   3/14/1998
  Description:  This script shows how comments can be used.
%END REM
For x = 1 to 10
  Print "x = " & x           ' print the loop index
Next x
```

' This comment was inserted to show use of apostrophes for commenting.

Example 2.22 shows both comment methods in action. You should notice that the apostrophe style of comments can be placed at the end of a line or on a separate line. The %REM...%END REM block is best suited for large comments that span multiple lines. Either method achieves the same result, but the %REM statements can be easier to spot than apostrophes.

2.8 Strings

Dealing with strings is a common occurrence. Fortunately, LotusScript provides numerous functions to handle strings. The following list shows a few.

- Mid
- UCase
- LCase
- Len
- Trim
- LTrim

- RTrim
- Left
- Right
- Instr

2.8.1 *Mid(String2Change, starting_position, optional_length) = String2GetCharacters*

The `Mid` function allows you to replace all or a portion of one string with characters from another.

- *String2Change* Designates the variable to alter.
- *starting_position* The character position within `String2Change` to use as the starting point for the replacement characters.
- *optional_length* Signals how many characters to retrieve from `String2GetCharacters` and place in the `String2Change` variable. If this parameter is omitted, all of `String2GetCharacters` is used.

Example 2.23

```
Dim string1 As String, string2 As String
string1 = "Today was a good day."
string2 = "slow day."
Print "Before: " & string1
Mid(string1,13,9) = string2
Print "After: " & string2
```

Output

Before Today was a good day.

After Today was a slow day.

Explanation

The system is directed to start at position thirteen within the string “Today was a good day.”, which is the `g` in `good`, and replace it with the first four characters from the string “slow day.” If the optional length parameter (4) had been left out, `string1` would contain “Today was a slow one.” after execution of the script.

2.8.2 *UCase/LCase(string)*

These two functions change the case of the string passed to them. `UCase` stands for uppercase and `LCase` is lowercase.

Example 2.24

```
Dim string1 As String, string2 As String
string1 = "Tony Patton"
```

```
Print LCase(string1)
Print UCase(string1)
```

Output

```
tony patton
TONY PATTON
```

Explanation

The two perform opposite functions. `LCase` changes every character in string to lowercase, and `UCase` converts every character to uppercase. These functions are excellent when checking for exact values entered by a user. We cannot guarantee the format of data entered, so converting it to lowercase or uppercase allows for easier comparisons.

Example 2.25

```
Dim answer As String
answer = InputBox("Please enter Yes or No.")
If (LCase(answer) = "no") Then
    MsgBox "No was entered."
ElseIf (UCase(answer) = "YES") Then
    MsgBox "Yes was entered."
End If
```

Explanation

The user is prompted to enter "yes" or "no." We cannot control the format of the input, so it is converted to lowercase to check for "no" and uppercase to check for "YES".

2.8.3 Len(String)

The `len` function returns the number of characters in the string passed into it.

Example 2.26

```
Dim entry As String
entry = InputBox("Please enter your name.")
If (Len(entry) < 1) Then
    MsgBox "You entered nothing!"
End If
```

Explanation

The user enters a string, and the `len` function is used to check for actual input.

2.8.4 Trim/LTrim/RTrim(String)

The `Trim` function removes all leading and trailing spaces from the string passed to it. `LTrim` removes leading spaces, and `RTrim` removes all trailing spaces.

Example 2.27

```
Dim string1 As String
```

```

string1 = "  This has spaces on both ends  "
Print Len(string1)
Print Len(Trim(string1))
Print Len(LTrim(string1))
Print Len(string1)
Print Len(RTrim(string1))

```

Output

```

32
28
32
30
32
30

```

Explanation

The `len` function is used to print the length of the variables with and without the `trim` functions. `Trim` removes the two spaces in the front and the end of `string1`. The `LTrim` function removes the two leading spaces from `string2`. The `RTrim` function removes the two trailing spaces from `string3`.

2.8.5 Left(String)/Right(String)

`Left` and `Right` are used to pull out a number of characters from a string.

Example 2.28

```

Dim answer As String
answer = InputBox$("Please enter yes or no.")
If (Left(UCase(answer),1) = "Y") Then
    MsgBox "You entered yes."
ElseIf (Right(LCase(answer),1) = "n") Then
    MsgBox "You entered no."
End If

```

Explanation

The `Left` and `Right` functions are used to check user input.

2.8.6 Instr(optional_starting_point, string1, string2, optional_compare_method)

The `Instr` function searches one string (`string1`) for the first occurrence of another string (`string2`). The `optional_starting_point` parameter is used to signal the point within the string (`string1`) to start the search. Also, the `optional_compare_method` is used to signal if a case-sensitive or insensitive search should be conducted (1 signals case-insensitive and the default, if no method is signaled, is 0 for case-sensitive). The `Instr`

function returns the position within string1 that string2 was found. It returns 0 if there is no match.

Example 2.29

```
Dim string1 As String, string2 As String
string1 = "Lotus Notes rules!"
string2 = "notes"
position = Instr(string1, string2, 1)
Print "The string was found at position " & position
```

Output

The string was found at position 7.

Explanation

The string “Lotus Notes rules!” was searched for “notes” starting at position one using a case-insensitive search. A match is found at position seven.

2.9 Format function

The `format` function allows a string, number, or date value to be specifically formatted for output. This is an excellent function to use when creating reports or displaying data that must have a certain appearance. The syntax of the function is:

```
Format(variable, optional_format)
```

The `optional_format` parameter controls how the variable will be formatted. Numbers can be displayed as a General Number, Currency, Fixed, Standard, or Percent. Date/Time values can be displayed as General Date, Long Date, Medium Date, Short Date, Long Time, Medium Time, or Short Time. LotusScript help gives more detailed descriptions of the options for the format parameter.

Example 2.30

```
Dim dateNow
dateNow = Now
MessageBox "Different date/time formats. " & Chr(10) & Chr(10) & _
    Format(dateNow, "General Date") & _
    Chr(10) & Format(dateNow, "Long Date") & Chr(10) & _
    Format(dateNow, "Medium Date") & Chr(10) & _
    Format(dateNow, "Short Date") & Chr(10) & Format(dateNow, "Long Time") & _
    Chr(10) & Format(dateNow, "Medium Time") & Chr(10) & _
    Format(dateNow, "Short Time")
```

Output



Figure 2.7 Format function

2.10 Predefined constants

Lotus Notes provides a set of constants with predefined names that can be used in your scripts. The constants are defined in a few files: `LSCONST.LSS`, `LSERR.LSS`, `LSXUIERR.LSS`, and `LSXBEERR.LSS`. The first file, `LSCONST.LSS`, contains constants that are used with various LotusScript functions/statements like `MessageBox`. The remaining three contain error messages. Each file is located in your Notes data directory and appendix A lists the contents of each.

Note The files mentioned should be located in the root Notes directory. For example, they are located in the `C:\Notes` directory on my laptop.

To take advantage of the files, your script(s) must be told to use them. This is accomplished with the `%INCLUDE` statement. This statement tells the system to insert the contents of an external file into the script. The insertion occurs at compile time. If the file to be included is not in the default Notes data directory, you must include its complete path.

Example 2.31

```
[Options]
%INCLUDE "LSCONST.LSS"
Sub Initialize
    MsgBox "This message was generated with constants.",MB_OK + _
        MB_ICONQUESTION,"Test"
End Sub
```

Explanation

This script takes advantage of the constants file. It uses predefined constants with the `MessageBox` statement instead of integer values that have to be used if the constant file has not been included.

2.11 UnderScore (continuation)

The underscore character (`_`) serves a special purpose in LotusScript. It allows a single statement to continue over multiple lines. The underscore character is the continuation character. Example 2.31 took advantage of it because the `MessageBox` statement had numerous lines.

2.12 Chapter summary

The purpose of this chapter was to build a foundation for the rest of the book. The syntax of the basic language constructs of LotusScript like looping, user interaction, and declaring variables is almost identical to Visual Basic, but not quite. This makes LotusScript a quick study for experienced Visual Basic developers, but they still have to learn the Notes/Domino environment. The topics covered in this chapter will be used extensively throughout the remainder of this book.

2.13 Chapter review

- Declaring variables
- Converting variables from one type to another
- The Options and Declarations section of a script.
- Looping with the `For`, `While`, `Do`, and `Forall` loops.
- Branching with `If` and `Select Case`.
- Displaying messages to the user with `MessageBox` and `Print` statements.
- Using the `InputBox` statement/function to get information from the user.
- Commenting scripts with the `%REM. . %END REM` block and apostrophe.
- Dealing with strings.
- Accessing predefined constants.