Function Fundamentals

by Alyce Watson

Table of Contents

What is a Function?

A function receives input arguments, performs some operations, and returns a value. Liberty BASIC has native functions and programmers may write their own functions. The input arguments may be literal values, variables, or expressions.

Literal Value: a hard-coded number or text string. Some examples of literal values:

```
"Hello World"
17
```

Variable: a name assigned by the program to hold string or numeric values that may vary over the course of program execution. Some examples of variables:

```
firstName$="John"
firstName$="Mary"
maxNum=45
maxNum=78
```

Expression: a set of numbers or characters that evaluate to a single value. Some examples of expressions:

```
"Hello " + "World"
"Hello " + UPPER$("World")
17 * 2 / 45
2 * (103 - 5)
```

Liberty BASIC Native Functions

Liberty BASIC has many native functions. Here is one:

```
Num = ABS(-6)
```

The ABS() function accepts a literal number, numeric variable, or numeric expression. It returns the absolute value of the number. Absolute value is the value without respect to sign. The numbers "6" and "-6" both have an absolute value of 6. The code above can be expressed in several different ways. Here are some of them:

```
Num = ABS(0-6)
Num = ABS(-6/1)
x = -1 * 6 : Num = ABS(x)
Num = ABS(7 + 2 - 15)
Num = ABS((8*0)-6)
```

In all cases, the value of Num is "6". That is the "return" from the function. The ABS() function takes a single numeric argument and returns a numeric value. The value can be assigned to a variable, as we did above. It can also be used in a larger expression, or it can be printed.

```
Total = 17 + ABS(-6)
Total = (17 + ABS(-6)) / 3.14
```

```
PRINT ABS(-6)
```

A function can itself be an argument in another function. The MAX() function returns the larger of two numeric arguments. Here it is, with the return from the ABS() function serving as one of the arguments.

```
BigNum = MAX( 17, ABS(-6))
```

Functions Can Return Strings

Functions with a "$" character appended to the function names return strings. One such function is the LOWER$() function. The return from the function must be used in places that expect a string. It cannot be used in a math expression, for instance.

```
greeting$ = LOWER$("HELLO")
print greeting$
```

produces
hello

The following example won't work, because it requires a numeric value:

```
'WRONG!
Total = 2 * LOWER$("HELLO")
```

Liberty BASIC Function Rules

The names of Liberty BASIC functions are case insensitive. That means that you can use ABS(), Abs(), or abs() and Liberty BASIC recognizes the function.

Liberty BASIC functions always expect at least one argument.

Liberty BASIC functions can accept a combination of string and numeric arguments. One such function is the MID$() function.

```
aVar$ = MID$("Hello",2,2)
```

User-Defined Functions

We aren't limited to the functions provided by Liberty BASIC. We can write our own functions. The general form of a function is as follows:

```
Function FunctionName(arg1, arg2,... argN)
    'some code here
    FunctionName=value
End Function
```

The function definition is signalled by the keyword **Function**. It is followed by the function's name, then a list of arguments separated by commas and enclosed in parentheses. The next lines contain whatever code routine is performed by the function. The return value is designated by assigning it to the name of the function. The function definition must end with **End Function**.

The Argument List

The arguments may be string, numeric, or a mixture of both. They may be literal values, variables, or expressions. The arguments must be separated by commas. These arguments are actually variables that are local to the function. They can be used in code contained in the function. When the function ends, the variables no longer have any value. The function can have no arguments, in which case the parentheses are empty. The parentheses are not optional.

Beginning with Liberty BASIC 4, arguments can be passed ByRef as well as by value. See: Passing Arguments into Subroutines and Functions - by value and BYREF

Scope

Functions have their own memory space that is separate from the memory of the main portion of the program and from other functions and called subroutines. Variables and branch labels that exist inside of the function do not exist in the main program. The main program cannot "see" them. Likewise, functions cannot "see" variables and branch labels that exist in the main program and in other functions and called subroutines. The part of the program that can "see" a variable is called its "scope."

Some entities have global scope. That is, they can be seen both inside of functions and in the main program. These things are global in scope: arrays, handles (window and control handles, file handles, DLL handles, communications ports), and structs. In Liberty BASIC, variables can be declared as GLOBAL, which means that they can be seen from all parts of a program.

Special global status is given to certain default variables used for sizing, positioning, and coloring windows and controls. These include variables WindowWidth, WindowHeight, UpperLeftX, UpperLeftY, ForegroundColor$, BackgroundColor$, ListboxColor$, TextboxColor$, ComboboxColor$, TexteditorColor$. The value of these variables can be seen in any function.

Calling User-Defined Functions

User-defined functions are used in the same way as native Liberty BASIC functions. You can use a function to assign a value to a variable, print the results of a function or use it as part of a larger expression. User-defined functions may return either string or numeric values. If a function is to return a string, the name must end with a "$" character.

StringFunction$()

NumericFunction()

The names given to the argument variables in the function definition represent variables that are local to the function. When the function is called, whatever literal value, expression or variable is contained within the commas will be evaluated and given the corresponding name in the argument list. See the following examples for a demonstration of the argument list. Notice in the last example, the variable names used to call the function are the same names used in the function, but in a different order. (They are name1$ and name2$.) This doesn't matter, because the argument variables are local to the function and are not the same variables as the ones with the exact same names used in the main program.

```
print LongestName$("Sam","Patricia")

first$="Carl"
last$="Gundel"
print LongestName$(last$,first$)

name1$="Liberty"
name2$="BASIC"
print LongestName$(name2$,name1$)
end

Function LongestName$(name1$,name2$)
    if len(name1$)>len(name2$) then
        LongestName$=name1$
    else
        LongestName$=name2$
    end if
End Function
```

Results in:

Patricia

Gundel

Liberty

User-Defined Function Rules

User-defined functions don't need input arguments. Native Liberty BASIC functions each require at least one argument, but user-defined functions do not have this restriction. If there are no arguments, the parentheses remain empty. The parentheses must not be omitted.

FunctionNameWithNoArguments()

User-defined function names are case sensitive, ,unlike native LB function names. GetName$() is not the same function as getname$().

User-defined functions may contain code that calls Liberty BASIC functions or other user-defined functions.

User-Defined Functions in Action

Here is a demo that changes the case of all words in a string so that the first letter is uppercase and the remaining letters are lowercase. Run the little demo to see how it works. Notice that the code inside the function uses several native Liberty BASIC string functions.

```
print NameCase$("hello world")
print NameCase$("CARL GUNDEL")
print NameCase$("tiTaNiC")
end

Function NameCase$(name$)
    nm$=word$(name$,1)
    while nm$<>""
        i = i + 1
        nm$=word$(name$,i)
        first$=upper$(mid$(nm$,1,1))
        rest$=lower$(mid$(nm$,2))
        new$=new$+" "+first$+rest$
    wend
    NameCase$=trim$(new$)
    End Function
```

Here is a simple numeric function that returns the average of three values.

```
print Average(15,88,20)
print Average(10,20,30)
end
Function Average(num1,num2,num3)
    Average=(num1+num2+num3)/3
    End function
```