Windows Batch Scripting: Functions

Functions are de facto way to reuse code in just about any procedural coding language. While DOS lacks a bona fide function keyword, you can fake it till you make it thanks to labels and the `CALL` keyword.

There are a few gotchas to pay attention to:

1. Your quasi functions need to be defined as labels at the bottom of your script.
2. The main logic of your script must have a `EXIT /B [errorcode]` statement. This keeps your main logic from falling through into your functions.

Defining a function

In this example, we'll implement a poor man's version of the *nix tee utility to write a message to both a file and the stdout stream. We'll use a variable global to the entire script, `%log%` in the function.

```
@ECHO OFF
SETLOCAL

:: script global variables
SET me=%~n0
SET log=%TEMP%\%me%.txt

:: The "main" logic of the script
IF EXIST "%log%" DELETE /Q %log% >NUL

:: do something cool, then log it
CALL :tee "%me%: Hello, world!"

:: force execution to quit at the end of the "main" logic
EXIT /B %ERRORLEVEL%
```

```
:: a function to write to a log file and write to stdout
:tee
ECHO %* >> "%log%"
ECHO %*
EXIT /B 0
```

Calling a function

We use the `CALL` keyword to invoke the quasi function labelled `:tee`. We can pass command line arguments just like we're calling another batch file.

We have to remember to `EXIT /B` keyword at the end our function. Sadly, there is no way to return anything other than an exit code.

Return values

The return value of `CALL` will always be the exit code of the function. Like any other invokation of an executable, the caller reads `%ERRORLEVEL%` to get the exit code.

You have to get creative to pass anything other than integer return codes. The function can `ECHO` to stdout, letting the caller decide to handle the output by pipeling the output as the input to another executable, redirecting to a file, or parsing via the `FOR` command.

The caller could also pass data by modifying a global variable, however, I try to avoid this approach.