All DOS versions interpret certain characters before executing a command.
Some well know examples are the percent sign (`%`), and the redirection symbols (`< | >`).

Windows 95/98 and NT, and OS/2 too, also interpret double quotes (`"`) and ampersands (`&`), as shown in the Conditional Execution page.

In batch files, the percent sign may be "escaped" by using a double percent sign (`%%`).
That way, a single percent sign will be used as literal within the command line, instead of being further interpreted.

In Windows 95/98 and NT, and OS/2 too, redirection symbols may be escaped by placing them between double quotes (`">"`).
However, the quotes themselves will be passed to the command too, unlike the double percent sign.

Windows NT and OS/2 also allow the use of carets (`^`) to escape special characters. Even linefeeds can be escaped this way, as is shown in the Useless Tips page.

If you intend to "nest" commands with escaped characters, you may need to escape the escape character itself too.
In general, that won't make it any easier to read or debug your batch files, however.

Since the introduction of delayed variable expansion a new challenge is to escape exclamation marks, the "delayed" version of the percent sign.
Unlike percent signs, that can be escaped by doubling them, exclamation marks cannot be escaped by adding an extra exclamation mark.
Nor does a caret before the exclamation mark work, unless quoted (i.e. `ECHO ^!` will fail to display an exclamation mark, whereas `ECHO "^!"` *will* display a *quoted* exclamation mark: `"!"`).
Jaime Ramos sent me this link (http://stackoverflow.com/questions/3288552/how-can-i-escape-an-exclamation-mark-in-cmd-scripts) where the solution can be found: use `^^!`.
The trick is that a single caret will be used to escape the exclamation mark in the first "pass" of command line interpretation, but delayed variable expansion adds a second "pass" where the exclamation mark will be interpreted. If you don't get it, never mind, just remember the double caret before the exclamation mark.

Using `FIND` to search files or standard output for a string containing doublequotes used to be a pain... I often used `FINDSTR` instead, in these cases.
Robert Cruz explained to me that *in this particular case only* you need to escape doublequotes by doublequotes, i.e. a single doublequote must be replaced by 2 doublequotes.
He also sent me this link (http://technet.microsoft.com/en-us/library/2ca66b22-3b7c-4166-8503-eb75fc53ab46) on the subject.

Summary

Escape Characters		
Character to be escaped	**Escape Sequence**	**Remark**
%	%%	
^	^^	May not always be required in doublequoted strings, but it won't hurt
&	^&	
<	^<	
>	^>	
|	^|	
'	^'	Required only in the FOR /F "subject" (i.e. between the parenthesis), *unless* backq is used
`	^`	Required only in the FOR /F "subject" (i.e. between the parenthesis), *if* backq is used
,	^,	Required only in the FOR /F "subject" (i.e. between the parenthesis), even in doublequoted strings
;	^;	
=	^=	
(	^(	
)	^)	
!	^^!	Required only when delayed variable expansion is active
"	""	Required only inside the search pattern of FIND
\	\\	Required only inside the regex pattern of FINDSTR
[	\[	
]	\]	
"	\"	
.	\.	
*	*	
?	\?	

Unexpected behaviour of CALLs:

Try this demo batch file, it shows some results you may not have expected.
Always test when escaped strings have to be passed on to CALLed commands.

```
@ECHO OFF
IF "%~1"=="" (
        CLS
        ECHO Demo script showing how CALL adds its own caret escape characters to arguments
        ECHO.
        ECHO Note the loss of the percent sign even when doublequoted
        ECHO Command         : ECHO "%%A^("
        ECHO Expected output : "%%A^("
        ECHO Actual output   : "%A^("
        ECHO.
        ECHO Command         : ECHO %%A^^(
        ECHO Expected output : A^(
        ECHO Actual output   : %A^(
        ECHO.
        ECHO Command: CALL :Subroutine "%%A^("
        CALL :Subroutine "%A^("
        ECHO Command: CALL "%%~f0" "%%A^("
        CALL "%~f0" "%A^("
        ECHO.
) ELSE (
        ECHO.
        ECHO Start of rerun, note the automatically added escape characters and loss of percent signs
        ECHO Command         : ECHO "%%~1"
        ECHO Expected output : "%%A^("
        ECHO Actual output   : "%~1"
        ECHO.
        ECHO Command: ECHO.%%~1
        ECHO Expected output : A(
        ECHO Actual output   : %~1
        ECHO End of rerun
        ECHO.
)
GOTO:EOF

:Subroutine
ECHO.
ECHO Start of Subroutine, note the automatically added escape characters and loss of percent signs
ECHO Command         : ECHO "%%~1"
ECHO Expected output : "%%A^("
ECHO Actual   output : "%~1"
ECHO.
ECHO Command         : ECHO.%%~1
ECHO Expected output : A(
ECHO Actual   output : %~1
ECHO End of Subroutine
ECHO.
GOTO:EOF
```

The script above will generate the output shown below:

```
Demo script showing how CALL adds its own caret escape characters to arguments

Note the loss of the percent sign even when doublequoted
Command         : ECHO "%A^("
Expected output : "%A^("
Actual output   : "A^("

Command         : ECHO %A^(
Expected output : A(
Actual output   : A(

Command: CALL :Subroutine "%A^("

Start of Subroutine, note the automatically added escape characters and loss of percent signs
Command         : ECHO "%~1"
Expected output : "%A^("
Actual   output : "A^^("

Command         : ECHO.%~1
Expected output : A(
Actual   output : A^(
End of Subroutine

Command: CALL "%~f0" "%A^("

Start of rerun, note the automatically added escape characters and loss of percent signs
Command         : ECHO "%~1"
Expected output : "%A^("
Actual output   : "A^^("

Command: ECHO.%~1
Expected output : A(
Actual output   : A^(
End of rerun
```