Printing Example:

Syntax:  \( \text{format t String Args} \)

Explanation:
'It prints the string and the arguments on the screen. It always return NIL at the end.

Example:  
\( > \text{(format t "Hello")} \)
Hello
NIL

\( > \text{(format t "The value of pi is: ~F" pi)} \)
The value of pi is 3.14159
NIL

\( > \text{(format t "~S divided by ~S~% results in ~F" 1.2 2.4 (/ 1.2 2.4))} \)
1.2 divided by 2.4
results in 0.500000
NIL

Explanation:

\~%: Insert a CR. This is like \n in a C printf call.
\~&: Insert a CR unless already at beginning of line.
\~S: S-expression. Insert an arbitrary Lisp expression, and print it just as that expression (would normally be printed if it were returned as a value).
\~A: ASCII. Similar to \~S, but if one of the args is a string, its double quotes will be dropped.
\~D: Integer (like %d in C)
\~widthD: extra spaces. (format NIL "~5D" 6) returns " 6"
\~D: commas. (format NIL "~;D" 1234567) returns "1,234,567"
\~F: Floating point (like %f in C)

Syntax:  \( \text{format NIL String Args} \)

Explanation:
'It prints the string and the arguments on the screen. It does not return NIL at the end.

Example:  
\( > \text{(format NIL "Hello")} \)
"Hello"

\( > \text{(format NIL "The value of pi is: ~F" pi)} \)
"The value of pi is 3.14159"

\( > \text{(format NIL "~S divided by ~S~% results in ~F" 1.2 2.4 (/ 1.2 2.4))} \)
"1.2 divided by 2.4
results in 0.500000"
Sorting Example:

Sorting a simple List:

```lisp
> (setq Test '(1 2 3 4 3 2 1))
(1 2 3 4 3 2 1)
> (setq Test (sort Test #'>))
(4 3 3 2 2 1 1)
> (setq Test (sort Test #'<))
(1 1 2 2 3 3 4)
```

Sorting a simple List:

```lisp
> (setq Test2 '((A 1) (B 2) (C 3) (B 2) (A 1)))
((A 1) (B 2) (C 3) (B 2) (A 1))
> (defun Val (Entry) "Return the value part of a (LETTER VALUE) pair" (second Entry))
VAL
> (defun Greater-Val? (Entry1 Entry2) "Is the Val part of Entry1 bigger than Val part of Entry2?" (> (Val Entry1) (Val Entry2)))
GREATER-VAL?
> (setq Test2 (sort Test2 #'Greater-Val?))
((C 3) (B 2) (B 2) (A 1) (A 1))
```

Recursion Example:

```lisp
> (defun factorial(x) (if (= x 0) 1 (* x (factorial (-x 1)))))
FACTORIAL
> (factorial 3)
6
```