Handy Logo By Example

Beeper, LCD Display, Loops, Numbers

Type the following examples into the Handy Logo Command Center. Checks (√) indicate most critical features.

√ beep
√ beep wait 2 beep
√ repeat 4 [beep wait 2]
√ loop [beep wait 2]
√ STOP

√ note 60 10
note 5 * 12 7 + 3
√ print 5 * 12
print [5 * 12]
print 2 + 3 * 4
print random 10
print random 2 + 10

Common errors:
• note 60 10
  note60 undefined  (Spaces are important!)
• note 60
  Not enough inputs to note
• note 60 wait 10
  wait doesn't output
• note 60 10 20
  You don't say what to do with 20
√ note 5*12 7 + 3
  5*12 undefined (Spaces are important!)

Exercise: Write a command that plays a random song

Motors

Plug motors into motor ports A and B on the Handy Board (labelled MOTOR-0 and MOTOR-1)

√ a, on
√ off
√ on wait 10 off
√ onfor 10
√ b, on wait 10 ab, toggle
√ STOP
√ a, repeat 6 [toggle wait 5]
√ repeat 4 [onfor 10 rd]
√ thisway on b, thatway on
√ STOP onfor 10

setpower 1 on
setpower 8

Exercise: Predict the state of the motors at the end of the following commands
• b, onfor 10 ab, toggle
• cd, on bc, rd c, toggle abcd, toggle
Digital Sensors (Switches)

Plug touch sensors (microswitches) into digital sensor ports 7 and 8. The following examples assume that you press STOP after every loop example to stop the loop.

√ loop [print switch 7]  
Prints 1 (true) when switch 7 on, 0 (false) when off.

√ a, on waituntil [switch 7] off  
Turns motor a on; pressing switch 7 turns off.

loop [waituntil [switch 7] onfor 10]  
Turns motor a on for a second every time switch 7 is pressed. Motor stays on if switch is held down.

Type long commands like the following in the Command Center without a line return!

loop [waituntil [switch 7] a, onfor 10  
waituntil [switch 8] b, onfor 10]  
Switches 7 and 8 turn on and b in alternation.

loop [waituntil [switch 7] on  
waituntil [switch 8] off]  
Switch 7 turns motor a on, switch 8 turns it off.

The following does not toggle motor a on and off. Why?

loop [waituntil [switch 7] on  
waituntil [switch 7] off]

√ loop [waituntil [not switch 7]  
waituntil [switch 7] toggle]  
Switch 7 toggles motor a on and off.

Example of edge-triggered action.

loop [if switch 7 [a, onfor 10]  
if switch 8 [b, onfor 10]]  
Switch 7 turns on a, switch 8 turns on b, any order.

Switch ignored when motor on.

√ loop [ifelse switch 7 [a, on] a  
[a, off]]  
a is on when switch 7 is pressed and off otherwise.

Example of level-triggered action.

Exercise: Predict the behavior of the following commands:

• a, on if switch 7 [toggle]
• a, on waituntil [switch 7] toggle
• a, on loop [if switch 7 [toggle]]
• a, on loop [waituntil [switch 7] toggle]
• a, on loop [waituntil [not switch 7] waituntil [switch 7] toggle]

Challenges: Write commands to implement the following behaviors:

• a is on when switch 7 is pressed and off otherwise; b is on when switch 8 is pressed and off otherwise.
• Switch 7 turns a on and b off, switch 8 turns a off and b on (in any order)
• Only one of a and b is on. Which one is on changes every time switch 7 is pressed.

Note: The following cannot be accomplished without mutable variables and/or concurrency:

• Switch 7 toggles motor a, switch 8 toggles motor b (in any order).
• Switch 7 toggles motor a, switch 8 reverses its direction.
• Switch 7 turns on a for a second, switch 8 turns on b for a second. Switches active even when motors on.
Analog Sensors

Plug light sensor into analog sensor ports 0. The following examples assume that you press STOP after every loop example to stop the loop.

\[ \text{\textit{loop \{print sensor 0 wait 1\}}} \] Continuously prints value of sensor 0 (0 -- 255). Typically, low value means sense "a lot"; high value means sense "a little".

Turn menu knob past menu item (7) to see analog display mode for all 7 analog sensors.

\[ \text{a, on wait until \{(sensor 0) > 100\} off} \] Turns off motor when light sensor blocked.

\[ \text{\textit{a, loop \{if else (sensor 0) < 100 \{on\} \{off\}\}}} \] Motor on in light, off in dark.

Common bugs:

\[ \checkmark \text{ sensor 0 < 100} \equiv \text{sensor (0 < 100)}; \text{want (sensor 0) < 100} \]

\[ \checkmark \text{ sensor 0 < sensor 1} \equiv \text{sensor (0 < sensor 1)}; \text{want (sensor 0) < (sensor 1)} \]

Procedures

Type the following procedures into the procedures window. Press the Download button to tell the Handy Board that there are new procedures.

\[ \text{\textit{to double-beep}} \]
\[ \text{beep wait 2 beep} \] Procedure begins with \textit{to}, ends with \textit{end}. After download, invoke via \textit{download} in Command Center.

\[ \text{\textit{to wiggle :num :tenths}} \]
\[ \text{repeat :num \{a, onfor :tenths rd\}} \] Parameter declarations and uses marked by colon. Sample invocation: \textit{wiggle 4 10}

\[ \text{\textit{to dark? :port ; analog port}} \]
\[ \text{output (sensor :port) > 100} \] Comments introduced with semi-colon. \textit{output} returns a result.

\[ \text{\textit{to find-light}} \]
\[ \text{forward 20} \]
\[ \text{if (or (dark? 0) (dark 1)) [find-light]} \] \textit{forward} defined below. (order is irrelevant). Handy Logo supports \texttt{and}, \texttt{or}, and \texttt{not} (bitwise). Tail recursion is an alternative to loops. (Non-tail recursion limited by tiny stack size.)

\[ \text{\textit{to find-light-loop}} \]
\[ \text{loop \{forward 20} \]
\[ \text{if (and (not (dark? 0)) (not (dark? 1))) [stop]} \] \textit{loop} to find light. \textit{stop} exits the current procedure.

You can put any Handy Logo commands (including procedure invocations) into the Menu Items boxes, and then run the Handy Board untethered from the computer. You can execute a menu item by either (1) selecting it with the menu knob and pressing the START button or (2) pressing the menu item number on a TV remote control.
Variables

```lisp
(global [count black])

to initialize
  setcount 0 setblack 100
end
to test-black
  if count < 10
    [forward 10
      if (sensor 0) > black
        [setcount count + 1]
      test-black]
  end
to average :s :times
  let [sum 0]
  repeat :times
    [make "sum :sum + (sensor :s)]
  output sum / :times
end
```

Concurrently

Concurrent can modularize subtasks that are unnecessarily intertwined with a single thread of control. Exercise: Based on the following, write procedures to solve problems in Note of digital sensor section.

```lisp
loop [waituntil [switch 7] a, onfor 10
  waituntil [switch 8] b, onfor 10] Switches 7 and 8 turn on a and b in alternation.

loop [switch 7] a, onfor 10
  waituntil [switch 8] b, onfor 10] Switch ignored when (1) motor on (2) not its "turn".

loop [waituntil [switch 7] a, onfor 10
  waituntil [switch 8] b, onfor 10] launch creates independent task (control thread)

loop [switch 7] toggle
  [waituntil [not switch 7]
    waituntil [switch 7]
    toggle]

loop [switch 7] [beep]
  when [switch 8]
  stoprules
end
```

Concurrent can modularize subtasks that are unnecessarily intertwined with a single thread of control. Exercise: Based on the following, write procedures to solve problems in Note of digital sensor section.