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**	The Handy Board beeps!
√ **beep wait 2 beep**	Beeps, waits 2 tenths of a second, & beeps again. **wait** grabs control!
√ **repeat 4 [beep wait 2]**	Repeats beep/wait/beep 4 times
√ **loop [beep wait 2]**	Repeats beep/wait/beep infinitely
√ STOP	press STOP button to stop loop (or any Handy Board program).
√ **note 60 10**	note *frequency tenths*; use frequencies in range 40 -- 120.
note 5 * 12 7 + 3	Can use arithmetic where numbers are expected.
√ **print 5 * 12**	Prints **60** on the LCD display.
print [5 * 12]	Prints **5 * 12** on the LCD display.
print 2 + 3 * 4	Parsed as **(2 + 3) * 4**; use parentheses to override default.
print random 10	Prints psuedo-random number between 0 and 9.
print random 2 + 10	Parsed as **random (2 + 10)**; use parentheses to override.

Common errors:

• **note60 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**	Makes **a** the current motor port and turns it on.
√ **off**	Turns the current motor (**a**) off.
√ **on wait 10 off**	Turns motor **a** on for one second.
√ **onfor 10**	Abbreviation for above commands; **onfor** grabs control!.
√ **b, on wait 10 ab, toggle**	Turns motor **b** on for one second, then turns **b** off and **a** on.
√ STOP	Press STOP button to stop.
√ **a, repeat 6 [toggle wait 5]**	Turns motor **a** on and off 3 times for half second intervals.
√ **repeat 4 [onfor 10 rd]**	Reverses direction of motor **a** every second for 4 times.
√ **thisway on b, thatway on**	Turns **a** on in green direction and **b** on in red direction.
√ STOP **onfor 10**	Press STOP to stop. After STOP, **a** is always the current motor.
setpower 1 on	Turns **a** on with low power; range is 0 (off) to 8 (full)
setpower 8	Sets **a** back to full power (This is the default after STOP.)

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 **a** and **b** in alternation. Switch ignored when (1) motor on (2) not its "turn".

```
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, 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.

√ `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.

`a, on waituntil [(sensor 0) > 100] off` Turns off motor when light sensor blocked.

√ `a, loop [ifelse (sensor 0) < 100 [on] [off]]` Motor on in light, off in dark.

Common bugs:

√ • `sensor 0 < 100` ≡ `sensor (0 < 100)`; want `(sensor 0) < 100`

√ • `sensor 0 < sensor 1` ≡ `sensor (0 < sensor 1)`; 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.

√
```
to double-beep
  beep wait 2 beep
end
```
Procedure begins with **to**, ends with **end**. After download, invoke via **download** in Command Center.

√
```
to wiggle :num :tenths
  repeat :num [a, onfor :tenths rd]
end
```
Parameter declarations and uses marked by colon. Sample invocation: **wiggle 4 10**

√
```
to dark? :port ; analog port
  output (sensor :port) > 100
end
```
Comments introduced with semi-colon. **ouptut** returns a result.

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

```
to forward :tenths
  ab, onfor :tenths
end
```

```
to find-light-loop
  loop [forward 20
        if (and (not (dark? 0))
                (not (dark? 1)))
           [stop]]
end
```
Procedure using **loop** to find light.

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

```
global [count black]
```
Declare global variables **count** and **black**.

```
to initialize
  setcount 0 setblack 100
end
```
Assign to global variable **X** via **setX newValue**.

```
to test-black
  if count < 10
     [forward 10
      if (sensor 0) > black
         [setcount count + 1]
      test-black]
end
```
Reference global variable **X** via **X**.

```
to average :s :times
  let [sum 0]
    repeat :times
     [make "sum :sum + (sensor :s)]
    output sum / :times
end
```
Declare local variable **sum**.

Assign to local variable **X** via **make "X newValue**.

Reference local variable **X** via **:X**.

Concurrency

Concurrency 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.

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

```
to enable-switches
  launch [loop [waituntil [switch 7]
                a, onfor 10]]
  launch [loop [waituntil [switch 8]
                b, onfor 10]]
end
```
launch creates independent task (control thread)
Both switches are active even when motors are on!

```
to wiggle-and-beep
  forever [a, onfor 2 rd]
  every 10 [beep]
end
```
forever [...] is sugar for **launch [loop [...]]**.
every time [action] performs **action** every **time** tenths of a second.

```
to toggle-task
  a, forever
      [waituntil [not switch 7]
       waituntil [switch 7]
       toggle]
end
```
Here creates a looping task with edge-triggered condition.

```
to wiggle-and-beep-when-bumped
  forever [a, onfor 2 rd]
  when [switch 7] [beep]
  waituntil [switch 8]
  stoprules
end
```
when is sugar for looping task with edge-triggered condition.

stoprules stops all members of a task family except current task. **launch**/START create a new family; **forever**, **every**, **when** add new task to current family.