```lisp
;;; Use command clisp to invoke common Lisp
;;;(load "LispCheatSheet") slurps in LispCheatSheet.cl
;;; to print this, use enscript -C -Eelisp -2r

(defun helloWorld ()
  ;; prints the standard message
  (print "Hello World"))

;; fun with predicates
(defun predicateDemo ()
  (print (member 3 (list 1 2 3 4 5 6)))
  (print (symbolp 'foo))
  (print (eq 4 (+ 2 2)))  ; good for atoms
  (print (equal '(1 2 3) '(1 2 3)))  ; good for other things
  (print (null ()))  ; should be TRUE
  (print (listp ()))  ; should be TRUE
  (print (listp '(1 2 3)))  ; should be TRUE
  (print (listp '3))  ; should be FALSE

)

(defparameter *aGlobalVar* 1)   ; note the earmuffs

;; ash is arithmetic shift
(defun ashDemo ()
  (print (list (ash 16 2) (ash 8 -1)))
)

;; (defun aFunction (possibly empty list of parameters)
;; <function body>
;; )

;; let creates local variables, e.g.
(defun letDemo ()
  (print (let ((a 6) (b 3)) (+ a b)))
)

;; labels creates local functions that can call each other, e.g.
(defun labelsDemo ()
  (print (labels ((a (n) (+ n 5))
          (b (n) (+ (a n) 6)))
    (b 10)))
)

;; an example of expt
(defun exptDemo ()
  (print (let ((pi 3.14) (e 2.7)) (list (expt pi e) (expt e pi))))
)

;; basic lisp manipulation
(defun listDemo()
  (print (cons 'a 'b)) ; creates a dotted pair
  (car '(this is a list))
  (cdr '(this is another list))
  (cons 'quick '(brown fox))
  (print (reverse '(1 2 3)))
  (print (member 3 (list 1 2 3 4 5 6)))
)

;; of course every language needs an if statement
(defun ifDemo()
  (print (if (= (+ 1 2) 3) 'yup 'nope))
)

;; the cond statement does more!
(defun condDemo()
  (let ((a 2))
    (cond ((eq a 1) (princ "a is 1"))
          ((eq a 3) (princ "a is 3"))
          (t        (princ "a is something else")))
    )
)

;; creating an association list
(defparameter *nodes* '((living-room (you are in a living room))
                        (garden (you are in a garden))
                        (attic (you are in the attic))))
(defun assocDemo()
  (print (assoc 'garden *nodes*))
)

;; example of mapcar
(defun mapcarDemo()
;  (mapcar #'sqrt '(1 2 3 4 5))
  (mapcar (function sqrt) '(1 2 3 4 5))
)

;; example of append
(defun appendDemo()
  (append '(1 2) '(3 4))
)

;; example of apply
(defun applyDemo()
  (apply #'append '((10 20) (30) (40 50)))
)

;; an example from Barski
(defun say-hello ()
  (princ "Please type your name:")
  (let ((name (read-line)))
    (princ "Nice to meet you, ")
    (princ name)))

;; use this function to run all the others
(defun allDemos()
  (when t
    (helloWorld) (predicateDemo) (listDemo)
    (ashDemo) (letDemo) (labelsDemo) (exptDemo) (ifDemo)
    (condDemo) (assocDemo) (mapcarDemo) (appendDemo)
    (applyDemo)
    t)
)

```