2)a)

(define (hyp x y)

(sqrt (+ (expt x 2) (expt y 2)))

)

b)

(define (segundo list)

(car (cdr list))

)

c)

(define (divsegura a b)

(if (eq? b 0) 0 (/ a b))

)

d)

(define (saludo x)

(string-append "Hola " x)

)

e)

(define (pedirNumero)

(display "Ingrese un numero: ")

(let ((x (read)))

(if (number? x)

x

(begin

(display "No es un numero, intente de nuevo.\n")

(pedirNumero)))))

3)a)

\* usa pedirNumero

(define (checkNumero)

(let ((x (pedirNumero)))

(cond

((< x 0) (display "El numero es menor a 0.\n"))

((> x 0) (display "El numero es mayor a 0.\n"))

(else (display "El numero es 0.\n")))))

b)

(define (sumaComprendidos)

(define (sumaValoresComprendidos x y)

(if (= x y)

y

(+ x (sumaValoresComprendidos (+ x 1) y))))

(let ((a (pedirNumero))

(b (pedirNumero)))

(if (< a b)

(display (sumaValoresComprendidos (add1 a) (sub1 b)))

(display (sumaValoresComprendidos (add1 b) (sub1 a)))

)

(newline)))

C)

(define (generarLista inicio fin)

(if (> inicio fin)

'()

(cons inicio (generarLista (+ inicio 1) fin))))

(define (crearLista)

(let ((x (pedirNumero)))

(if (= x 1)

(list 1)

(append (generarLista 1 x) '()))))

D)

(define (generarListaInversa inicio fin)

(if (< fin inicio)

'()

(cons fin (generarListaInversa inicio (- fin 1)))))

(define (crearLista)

(let ((x (pedirNumero)))

(cond

((< x 1)

(begin

(display "Ingrese un numero mayor a 0. \n")

(crearLista)))

((= x 1)

(list 1))

(else

(generarListaInversa 1 x)))))

E)

(define (pedirLista)

(display "Ingrese una lista: ")

(let ((x (read)))

(if (list? x)

x

(begin

(display "No es una lista, intente de nuevo.\n")

(pedirLista)))))

(define (crearLista)

(let ((x (pedirLista)))

(cond

((null? x) '())

(else

(append (crearListaAux (cdr x)) (list (car x)))))))

(define (crearListaAux lst)

(if (null? lst)

'()

(append (crearListaAux (cdr lst)) (list (car lst)))))

F)

(define (pedirLista)

(display "Ingrese una lista: ")

(let ((x (read)))

(if (list? x)

x

(begin

(display "No es una lista, intente de nuevo.\n")

(pedirLista)))))

(define (cuentaAtomos)

(let ((x (pedirLista)))

(define (contar lst)

(cond

((null? lst) 0)

((list? (car lst)) ; esto es por si esta anidada otra lista

(+ (contar (car lst)) (contar (cdr lst)))) ; aca sumamos la lista anidada

(else

(+ 1 (contar (cdr lst)))))) ; si no sumo un elemento

(contar x)))

G)

(require(lib "compat.ss"))

(define (pedirAtomo)

(display "Ingrese un átomo: ")

(let ((x (read)))

(if (atom? x)

x

(begin

(display "No es un átomo, intente de nuevo.\n")

(pedirAtomo)))))

(define (pedirLista)

(display "Ingrese una lista: ")

(let ((x (read)))

(if (list? x)

x

(begin

(display "No es una lista, intente de nuevo.\n")

(pedirLista)))))

(define (mem?)

(let ((x (pedirAtomo))

(y (pedirLista)))

(define (buscar a lst)

(cond

((null? lst) #f) ; si esta vacia la lista -> falso

((list? (car lst)) ; si es lista, busca dentro de la misma para chequear anidamientos

(or (buscar a (car lst)) (buscar a (cdr lst))))

((equal? a (car lst)) #t) ; encuentra -> verdadero

(else (buscar a (cdr lst))))) ; continua con el resto de la lista en otro caso

(buscar x y)))