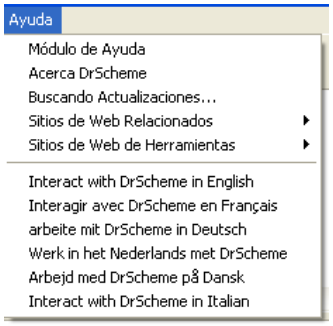
PRÁCTICO 10: Paradigma de programación funcional

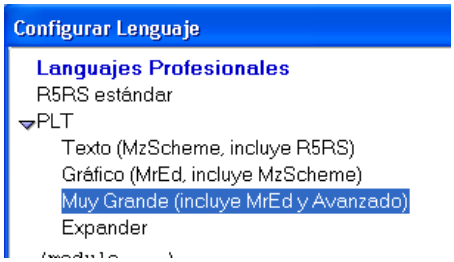
Actividades

Programar en Scheme

Para comenzar a trabajar con Dr. Scheme, una vez que haya ingresado, se puede elegir el idioma y se debe elegir el nivel del lenguaje con el que trabajará.

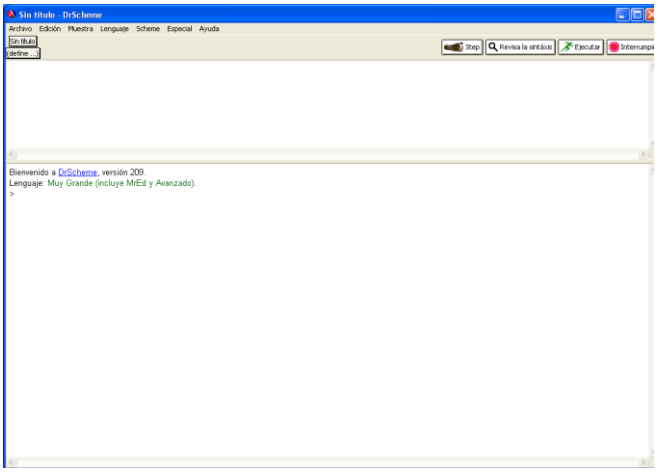
Para elegir el lenguaje, ir al menú LENGUAJE / ESCOGER LENGUAJE... y en la lista desplegable elegir MUY GRANDE.





1. Escriba las siguientes funciones en la mitad inferior de la pantalla.

➧ Recuerde que oprimiendo <ESC> + <p> repite la última línea ingresada




```
(add1 5)
(sub1 0)
(abs (add1 -5))
(- 6 9)
(quotient 17 5)
(/ 17 5)
(* 10 12)
(- (* 10 2) (+ 13 3))
(modulo 5 2)
(+ 1 2 3 4 5)
(* 1 2 3 4 5)
(max 2 12 3 10)
(min (* 4 6) (+ 4 6) (- 4 6))
(expt 2 5)
(expt 5 -2)
(sqrt 25)
(sqrt 2)
(negative? -6)
(zero? 44)
(positive? -33)
(number? 5)
(integer? 3.7)
(odd? 5)
(real? 82)
(even? 37)
(= 6 2)
(< 0.5 0.1)
(>= 3 30)
(<= -5 -3)
(> 6 2)
(eq? 5 5)
(define a 5)
(define b 3)
a
```

```
(+ a b)
(+ a c)
(* a (add1 b))

(define pi 3.1416)
(define (circ_sup r) (* pi (expt r 2)))
```
Utilice (circ_sup 2) para probarlo

```
(define (suptrap h b1 b2)(/ (* (+ b1 b2) h) 2) )
```
Utilice (suptrap 10 4 6) para probarlo

- Cuando el intérprete de Scheme evalúa una lista, espera que el primer ítem sea una expresión que representa una función y el resto de los elementos los considera los argumentos de la función. Para inhabilitar este comportamiento se utiliza la operación quote que se puede indicar también usando ' (comilla simple). Vea las diferentes salidas que se obtienen cuando se ingresa:

```
a
(a)
(quote a)
'a
'(a b c)
(quote (a b c))
(car (a b c))
(car '(a b c))
(cdr '(a b c))
(car(cdr '(a b c)))
(cdr '((a) (b) (c)))
(cons 'a '(q e r y))
(cons '(a z x e) '(q e r y))
(cons '(3) '(a b c))
(list 5)
(list 'a 'b 'c)
(list 'a '(b c) 'd)
(append '(1 2 3) '(a b c))
(append '3 '(a b c))
(append '(3) '(a b c))
(display 'x)
(display '(a b c))
(display "Hola rubia!")
(display (Hola rubia!))
(let((n '(4 5)))(display n))
(null? '( ))
(null? '(( )))
(eq? 'x 'x)
(eq? '(x z) '(x z))
(list? '(a d))
(list? 'a)
(if (eq? 5 7) 0 1)
(empty? '())
(empty? '(a b c))
```

- Algunas funciones sólo están disponibles invocando librerías, por ejemplo la sentencia: (require(lib”compat.ss”)) habilita:

```
(atom? 'x)
(atom? '(x))
(atom? 5)
(atom? x)
```

- Cuando escriba código, un comentario se debe encabezar por ; (punto y coma)

2. Escriba las siguientes funciones en la parte superior de la ventana y haga click sobre el botón Execute cuando termine. Para ejecutarlas escriba la llamada a la función en la parte inferior de la ventana.

   a) hyp que calcule la hipotenusa teniendo los lados a y b de un triángulo rectángulo.
   b) segundo que obtenga el segundo elemento de una lista (utilice car y cdr).
   c) div_segura que devuelva 0 si el divisor es 0 (utilice if).
   d) Pregunte por su nombre y le conteste Hola nombre!.
   e) Pregunte por un número y en caso de ingresar otro cosa, siga preguntando hasta que el valor ingresado sea un número.

➧ Para realizar la traza de ejecución deberá escribir: (require(lib“trace.ss“))

3. Escriba la función que reciba:

   a) un número y determine si es positivo, negativo o nulo informando con estas palabras el resultado. Incluya la lectura de datos.

   A partir de este punto realice la traza de ejecución, para ello invoque la librería correspondiente y escriba en el prompt (trace nombre_funcion)

   b) dos números y realice la suma de los números comprendidos entre ellos.
   c) un entero n y produzca una lista con los números desde 1 a n en orden ascendente.
   d) ídem al anterior pero en orden descendente.
   e) una lista simple como parámetro y retorne la lista inversa (no usar reverse).
   f) una lista y cuente la cantidad de átomos.
   g) un átomo y una lista e informe con verdadero o falso si el átomo está incluido en la lista, llame a esta función mem?
   h) dos listas simples y devuelva la unión de las dos listas. Utilice la función mem? para evitar átomos repetidos.
   i) un átomo y una lista y devuelva una lista donde todas las ocurrencias del átomo fueron quitadas (sin importar el nivel de profundidad de ocurrencia del átomo en la lista).

4. ¿Qué hacen las siguientes funciones Scheme?

```
(define (prueba atom lis)
  (cond
    ((null? lis) ())
     ((eq? atom (car lis)) (cons (car lis) (prueba atom (cdr lis))))
        (else (prueba atom (cdr lis)))
  )
)
```

Probar con (prueba 'a '(d a b a))

```
(require(lib"compat.ss"))
(define (que_hace lis)
   (cond
      ((empty? lis) 0)
      ((atom? (car lis))
      (cond
        ((eq? (car lis) null) (que_hace (cdr lis)))
              (else (add1 (que_hace (cdr lis))))
         )
      )
    (else (+ (que_hace (car lis)) (que_hace (cdr lis))))
  )
   )
```

Probar con (que_hace '(1 2 3 4 5 (a b c) 6 ()))

5. Considere la siguiente función:

```
(define (misterio ls)
        (if (null? (cddr ls))  (cons (car ls) ())
            (cons (car ls) (misterio (cdr ls))))))
```

a) ¿Qué devuelve (misterio '(1 2 3 4 5)) ?
b) Describa el comportamiento general de misterio.
c) Sugiera un buen nombre para la función misterio.

6. Escriba una función suma_todo que encuentre la suma de los números de una lista que puede contener sublistas anidadas de números. Pruebe con los siguientes ejemplos:

(suma_todo '((1 3) (5 7) (9 11))) → 36
(suma_todo '(1 (3 (5 (7 (9)))))) → 25
(suma_todo '()) → 0

7. Defina una función producto_punto, que tome como entrada 2 listas de la misma longitud, multiplique sus componentes correspondientes y sume los productos resultantes. Pruebe con los siguientes ejemplos:

(producto_punto '(3 4 -1) '(1 -2 -3)) → -2
(producto_punto '(0.3 0.5) '(8 2)) → 3.4
(producto_punto '() '()) → 0