# INTRODUCCIÓN

La presente guía servirá de soporte a los estudiantes del curso de fundamentos de Programación.

La guía viene con variados ejemplos de cada tema y al final de cada capítulo se presenta una sección de ejercicios de cada tema.

El estudiante está en la obligación de leer las guías antes de cada clase y resolver los ejercicios al final de cada tema visto en clase.

## CONCEPTOS MATEMATICOS BASICOS

Son los elementos necesarios que debemos aprender para desarrollar programa.

- Identificadores:
- Tipos de Datos
- Operadores y operandos
- Expresiones

**IDENTIFICADORES:** Representa los datos de un programa. Secuencia de caracteres que permite identificar una posición en la memoria de la computadora

Los identificadores valores asociados a un nombre. Se clasifican en:

**Constantes:** Datos Numéricos o alfanuméricos que no cambian durante la ejecución de un programa ejemplo pi=3.14, E=2.718228

**Variables:** Permite almacenar temporalmente un dato durante la ejecución de un proceso. Por ejemplo x=2, x=x+3, *x* inicialmente vale 2 y luego su valor se incrementa en 3, tomando *x* el nuevo valor de 5, Como vemos el valor de x puede variar, de allí el nombre de variable.

Los valores que se le asignan a una variable solo pueden tomar valores de ese tipo.
Los identificadores o sea los nombres que se le dan a las variables o constantes, elegidas para el algoritmo o programa deben ser significativos y tener relación con el objeto que representan por ejemplo:

*NombrePerson*: puede ser el nombre de una variable para representar nombres de personas.
*Precios*: para representar precios de diferentes artículos
*Notas:* para representar notas de una clase.

**Ejemplo**
*LongitudCircunferencia*= 2*Pi * *radio*

Pi: es una constante. Quiere decir que es un valor fijo (3.14)
*LongitudCircunferencia* y *radio* son variables y sus valores pueden cambiar

## TIPOS DE DATOS

Naturaleza del conjunto de valores que puede tomar una variable o constante. Los tipos de datos se clasifican en:

### Simples

a) **Numéricos**: Permite representar valores escalares en forma numérica
Ejemplo 4 2,5 3,14

b) **Lógicos**: Solo pueden tener dos valores cierto o falso. Representa el resultado de una comparación entre otros datos (Numéricos o Alfanuméricos)
Ejemplo la comparación (5>3) es Verdadera y la expresión (1>2) es falsa

c) **Alfanuméricos** : (string): Secuencia de caracteres alfanuméricos, que permite representar valores identificables de forma descriptiva, esto incluye nombres de personas, direcciones etc, Se representan encerrados entre comillas.
Ejemplos "calle 3 norte" "Juan Carlos López"

d) **Simbolo** (symbol) Secuencia de caracteres alfanuméricos que no pueden llevar espacios en blanco, van antecedido de una comilla simple.

Ejemplos 'rojo 'calle_3 'Sandra 'Juan_Carlos_Lopez

### Estructuras definida por el programador

a) Arreglos: (vectores o matrices)
b) Archivos
c) Apuntadores
a) Estructuras
b) Listas
c) Árboles

## OPERADORES Y OPERANDOS

Elementos que relacionan de forma diferente dos valores de una o más variables y/o constantes.

**Aritméticos**: Permiten la realización de operaciones matemáticas con los valores (variables y constantes)

| EXPRESION | | RESULTADO |
|---|---|---|
| 7/2 | = | 3.5 |
| 12 mod 7 | = | 5 |
| 4+2x5 | = | 14 |

En las expresiones anteriores ***/*** **mod** (en Scheme se representa como remainder) **+** y **x** corresponden a cada uno de los operadores.
7 2 12 4 5 corresponden a los operandos.

**EXPRESIONES** :

Son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones. Ejemplos

*A*+ (*b* +3)/*c*
*X*= *v*+ l

Según el tipo de datos que manipulan estas pueden ser:

a) Aritméticas
b )Relaciónales
d) Lógicas

**PRIORIDAD DE LOS OPERADORES ARITMETICOS**

- Todas las expresiones encerradas entre paréntesis se evalúan primero.
- Las expresiones con paréntesis anidados se evalúan de adentro hacia afuera, y los paréntesis más interno se evalúan primero.
- Dentro de una misma expresión los operadores se evalúan en el siguiente orden:

1 ∧ : (exponenciación)

2. **mod** (modulo o residuo de la division entre dos enteros),
**div** (cociente de la división entre dos enteros)
**x (**multiplicación),
/ (división)

**+** (suma) - (resta)

- Los operadores en una misma expresión con igual nivel de prioridad se evalúan de izquierda a derecha. ejemplos

1) 4+2*5=
4+10=
14

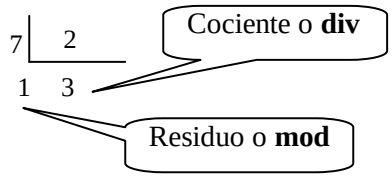
2) 23*2/5=
46/5=
9.2

2) 3+5*(10- (2+4))=
3+5*(10- 6)=
3+5*4=
3+20=
23

3) 2.1*(1.5+3.0*4.1)=
2.1*(1.5+12.3)=
2.1*13.8=
28.98

4) 3.5+5.09 - 3.5=
3.5+5.09-3.5=
8.59-3.5=
5.09

el **mod** es el residuo de la división entre **dos números enteros** ejemplo 7 mod 2 es igual a 1.




En caso de que los números no sean enteros no se les puede aplicar el mod por ejemplo 6,7 mod 2 seria un **error**!

**Operadores Relaciónales:** Establecen una relación entre dos valores.

Compara estos valores entre si y esta comparación produce un resultado de certeza o falsedad (verdadero o falso), tiene menor prioridad que los aritméticos.

Mayor que **>**
Menor que **<**
Mayor o igual que **>=**
Menor o igual que **<=**
Diferente **<>**
Igual **=**

**Ejemplos**

Si *a*=10 *b*=20 *c*=30

1. *a*<*b* es Verdadero
2) *c*>*a* es Verdadero

3) *a* + *b* > *c*
10+20>30
30 >30
es Falso

4) *a* - *b*< *c*
10 - 20 < 30
-10 < 30
es Verdadero

5) *a* * *b* <> *c*
10*20<>30
200<>30 es Verdadero

**Operadores Lógicos**: Se utilizan para establecer relaciones entre valores lógicos.

**And** y
**Or** o
**Not** Negación

**TABLA LOGICA DEL OPERADOR AND**

| operando 1 | operador | operando 2 | resultado |
|---|---|---|---|
| V | **AND** | V | V |
| V | | F | F |
| F | | V | F |
| F | | F | F |

V: Verdadero
F: Falso

**TABLA LOGICA DEL OPERADOR OR**

| operando 1 | operador | operando 2 | resultado |
|---|---|---|---|
| V | **OR** | V | V |
| V | | F | V |
| F | | V | V |
| F | | F | F |

**TABLA LOGICA DEL OPERADOR NOT**

| operando 1 | resultado |
|---|---|
| V | F |
| F | V |

**Ejemplos**

1) Si *a*=10, *b*=20 y *c*=30

1) (*a*<*b*) and (*b*< *c*)
(10<20) and (20 < 30)
V and V
V

2) Si *a*=10 *b*=12 *c* =13 *d* =10

((*a* > *b*) or (*a* < *c*)) and ((*a* = *c*) or (*a* >= *b*))
((10>12) or (10<13)) and ((10=13) or (10>=12))
( F or V ) and ( F or F )
V and F
F

3) Not (a=c) and (c>b)
Not (10=13) and (13 >12)
F and V
V

## TALLER 1

1. Responda Cual es la diferencia entra variable y constante.

2. Para las siguientes variables según el valor que contienen diga a que tipo pertenecen:

a) x=12.4
b) b) s=”Hola mundo”
c) cadena=’hola
d) placa=”3456”
e) color=’rojo
f) afirmacion=true
g) f=false

3. Diga que Variables y de qué tipo utilizaría para representar la siguiente información:

a) Dirección de una casa
b) Número telefónico
c) Nombre del país
d) Numero de cedula
e) Nombre de animal
f) Color de una fruta

4. Si a=15 b=35 c=40, Resuelva las siguientes expresiones y diga que valor toman

a) (a>b) or (a>c)
b) not( (a+b)>c)
c) ((*a* > *b*) or (*a* < *c*)) and ((*a* = *c*) or (*a* >= *b*))

## NOTACION PREFIJA

Antes de empezar a programar en Scheme debemos a aprender la notación prefija ya que en este lenguaje todas las expresiones matemáticas se escriben en esta notación.

En las expresiones prefijo todas las expresiones aritméticas están en paréntesis.

En la notación prefija primero se especifica el operador y luego los operandos, separados por espacios.

### Paso de Notación Infija a Prefija

### Ejemplo

Pasar a notación prefija las siguientes expresiones:

a) 2+8*5+6
2+(8*5)+6

1. Si no esta en paréntesis la expresión infija y hay operadores con diferente prioridad, primero se debe agrupar en paréntesis la expresión antes de pasarla a prefijo.

(+ 2 (* 8 5)) + 6
(+ (+ 2 (* 8 5) ) 6)

2 .Luego recorro la expresión de izquierda a derecha, ccuando se encuentra un operador abro paréntesis y escribo el operador seguido del operando izquierdo, seguido del operando derecho y finalmente se cierra el paréntesis. Repito el proceso hasta que toda la expresión este en prefijo

### Otra forma de resolver la expresión anterior

2+8*5+6
2+(8*5)+6
(+ 2 (* 8 5) 6)

Se siguen los mismos pasos pero si hay un mismo operador que opere sobre varios operandos en un mismo nivel de prioridad, entonces al pasar a prefijo se pueden agrupar en un solo paréntesis

b) 3+ (2 - 5)
(+ 3 (2 - 5))
(+ 3 (- 2 5))

c) (1 – 8 ) / (10 - 100)
(- 1 8) / (- 10 100 )

(/ (- 1 8) (- 10 100))

d) 1+2 - 3+4
(+ 1 2) – 3 + 4
(- (+ 1 2) 3) +4
(+ (- (+ 1 2) 3) 4)9

e) $5x^3 + 4x^2 + 2x - 4$
(* 5 x x x) + (* 4 x x) + (* 2 x) –4
(+ (* 5x x x) (* 4 x x ) (* 2 x)) – 4
(- (+ (* 5 x x x) (* 4 x x) (* 2 x)) 4)

**Resolver Expresiones en Prefijo**

**Ejemplos**:

1. Resolvamos las siguientes expresiones en prefijo

a) (- (* 5 7) (+ (* 7 8) 8))
(- 35 (+ 56 8))
(- 35 64)= -29

Resolvemos de izquierda a derecha y vamos resolviendo primero los paréntesis más internos.

b) (* 5 73 (+ 1 6))
(* 365 7)= 2555

c) (* (* 2 4) (+ 8 -9) 7)
(* 8 –1 7)= - 56

Cuando resuelvo lo que hay dentro del paréntesis, no se vuelve a escribir el paréntesis solo el resultado de la expresión

d) (* 6 8 (+ 100 –20))
(* 6 8 80)=3840

e) (- (* 18 3) –5)
(- 54 -5) = -59

f) (/ (+ (* 52 4) 6) 2)
(/ (+ 208 6) 2)
(/ 214 2)
107

g) (* (- (/ 98 3) 5) 4)
(* (- 32.6 5) 4)
(* 27.6 4)=110.4

h) (sqrt (+ (* 32 4) 16))
(sqrt (+ 128 16))
(sqrt 144)=12

remainder, obtiene el residuo de la división entre números enteros

i) (remainder (* 6 4) (/ 12 5))
(remainder 24 2.4) !error

j) (>5 (* 10 4))
(> 5 40)
falso

k) (< (+ 8 2) 20)
(< 10 20)
verdadero

i) (not (= 2 3))
(not falso)
verdadero

2. Cual es el resultado de las siguientes expresiones cuando x=4 x=2 x=7/2

a) (> x 3)

(> 4 3)= verdadero

(> 2 3) =verdadero

(> 7/2 3)=verdadero

b) (and (> 4 x) (> x 3))

(and (> 4 4) (> 4 3))
(and verdadero falso)
verdadero

(and (> 4 7/2) (> 7/2 3))
(and verdadero verdadero)= verdadero

(= (* x x) x))
(= (* 4 4) 4))=
(= 16 4)=falso

## TALLER 2 NOTACIÓN PREFIJA

1. Que valor devolverán las siguientes operaciones:

a) ( * (+ (- 5 2) (* 4 2) 7) (+ 5 5) 2)
b) (-(+ 7 3) (+ 4 6) (+ 10 10) 20)
c) (* ( – 10 –5) (+ 35 15))
e) (/ (+ (*(/ (+ 5 5) 2) (+ 2 2)) 10) 10)
f) (-(+(*(*(/ 40 2) 3) 2 20) 100) 40)

2. Representar las siguientes expresiones en notación prefija y resolver en Scheme:

a) [ (25 + 5) * 2] - 60
b) –50 * [(-50 + -15) * (-80 + 20)]
c) –2 * [(-5 + -15) + (-80 + 90)]
d) –5 * [(-10 – 20) – (-80 + 90)]
e) [(2 * 5) + [(4 – 2) + 8]]
f) 3+ (2 – 5 )
g) (1 - 18) / (11 – 2000)
h) $\sqrt{\frac{x}{4}} * \log x + 5 * x^3$
i) $\sqrt{a^3} - \sqrt{b^3}$

3 Que respuestas devuelven las siguientes expresiones
a) (-(+ 7 3) (+ 4 6) (+ 10 10) 20)
b) (+ (* 2 (- 82 67.5)) 100)
c) (/ (+(* 52 4) 6)2)
d) (* (- (/ 98 3)5) 4)
e) (sqrt (+ (* 32 4) 16))
f) (remainder (* 6 4 ) (/ 12 5))

4 Determine si es verdadero (true ) o falso (false)

a) (>8 -5)
b) (>= (+ 100 (+ 5 8 ) (* 10 1)) 2)
c) (< 4850 (+ 15 8 (+ (+ 90 9) (- 200 8)) (* 9 20)))
d) (> 5 ( * 10 4))
e) (< (+ 8 2) 20)
f) (and (> 4 3) (<=10 100))
g) (or (> 4 3) (=10 100))
h) (not (= 2 3))

5 Cual es el resultado de:

a) (> x 3)

b) (and (> 4 *x*) (> *x* 3) )

c) (= ( * *x x ) x*)

para a) *x=9* b) *x* = 3 c) *x* = 15/ 2

## CODIFICACIÓN

Desde mucho tiempo atrás, las maquinas han minimizado el esfuerzo humano, en procesos tediosos repetitivos. Los programas surgen como respuesta inmediata a la idea de construir una maquina y hacer que esta lleve a cabo un conjunto de instrucciones.

Para que la maquina entienda los algoritmos y los procese se necesitan que estén escritos en un lenguaje propio de la maquina o sea un lenguaje de programación.

Existen muchos lenguaje de programación, así como existen diferentes idiomas en el mundo como el ingles, español francés etc, con los cuales las personas se comunican, así también las maquinas tiene muchos idiomas para comunicarse java, C++, Visual Basic, Scheme y cada uno de ellos tiene sus propias características.

En el curso aprenderemos acerca del lenguaje de programación Scheme que es un lenguaje de programación Funcional.

Cuando se codifica un algoritmo en Scheme las expresiones se escriben en Notación Prefija. (que ya aprendimos), también el programa debe llevar cierta documentación esto son comentarios que se le hacen al programa para que sea más fácil de entender, tanto para nosotros como para otros programadores.

## PROGRAMACION FUNCIONAL

Un lenguaje funcional es aquel cuyos programas están basados en funciones, que son llamadas desde otras funciones, en estos lenguajes no existen formas explicitas de influir en el control del programa y la única forma de iterar es a través de la recursividad.

Estos lenguajes son especialmente útiles para representar problemas de modelación y simulación en las distintas áreas de la ingeniería. También es muy usado en el área de la inteligencia artificial

## Características de la Programación Funcional

Scheme nace a partir de list, sin embargo permite otras características como la asignación y por eso se le conoce como un lenguaje funcional impuro.

El valor de una expresión depende solo de los valores de sus subexpresiones, si las tiene.

Tiene una sintaxis sencilla, y muy poderosa.

Se basa en la evaluación de expresiones en lugar de la ejecución de instrucciones.

Permite la reutilización de código.

Datos fuertemente tipados: No acepta usar una variable en una función si sus datos no coinciden

## DEFINICIÓN DE VARIABLES EN SCHEM

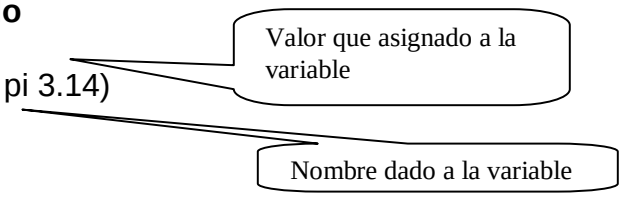
Se define una variable cuando queremos guardar un valor para que sea usada en cualquier parte del programa.

La definición de una variable tiene la siguiente sintaxis:

(define nombreVariable valorVariable)

**Ejemplo**

(define pi 3.14)




Cada vez que mencionemos la variable pi en Scheme el programa automáticamente reemplazara su valor.

## PROGRAMANDO EN SCHEME

Cuando hacemos un programa en Scheme debemos tener en cuenta realizar ciertos pasos:

**Propósito:** donde se describir que hace la función.

El **contrato** donde se indica sus datos de entrada y salida y el nombre dado a la función, para nuestro curso manejaremos esto como el **análisis de datos.**

También se piensan en algunos **Ejemplos**: es decir para ciertos valores de entrada que valores de salida se esperan obtener. (Este paso no lo usaremos en nuestro curso).

Luego se hacen las instrucciones propias del programa o sea el **cuerpo del programa** como decir el algoritmo pero escrito en lenguaje Scheme**.**

Finalmente se realiza la **prueba** es decir la **ejecución** del programa donde se dan valores a los datos o variables de entrada para obtener los resultados de salida. Este paso es tan importante como el cuerpo del programa, ya que en el cuerpo del programa se crea el programa, pero es en su ejecución donde realmente se utiliza. Si no se realiza este paso es como si el programa no se hiciera.

**Importante**: Los comentarios deben ir antecedidos de un punto y coma **;** para que estos no sean leídos por la maquina. (Solo los debemos ver nosotros).

Hagamos una función para sumar dos números

;**Propósito**: la función permitirá sumar dos números

**;analisis**
**Nombre de programa**: suma
**Datos de entrada** x de tipo numérico
y de tipo numérico

Tipo de los datos de entrada

**Tipo de datos de salida:** numérico

En Scheme no necesitamos darle nombre a los datos de salida

Una vez tenemos claro estos paso hacemos el cuerpo del programa
Bueno para escribir una función en Scheme tengamos en cuenta las siguientes cosas:

En Scheme todo va en paréntesis, cada paréntesis que abre debe cerrarse, para iniciar un programa utilizamos la palabra *define* para indicar que inicia el programa. Luego se abre un paréntesis en el cual se encierra el nombre que le damos a la función, para darle el nombre pensemos en cual es el tipo de dato de salida, esos nos da una idea de cómo podemos llamar a la función. Seguido al

nombre del programa vienen todas las variables o datos de entrada. Y luego viene la formula que resuelve el problema que debe estar escrita en prefijo.

**Forma general de de un programa en Scheme**

(define (nombreFuncion datosEntrada) (formulaPrefijo))

Entonces el algoritmo anterior en Scheme queda así:

```
;Cuerpo del programa
(define (suma x y)
(+ x y))
```

Observe que esta instrucción no va antecedida por ; (punto y coma), ya que se necesita que sea leída por el computador

Bueno ya tenemos el programa pero para que la maquina lo ejecute es necesario hacerle una llamada es decir indicarle a la maquina que corra el programa con los datos de entrada que queramos, por lo general esto se hace con los valores utilizados en el ejemplo (si lleva este paso). A esto lo llamamos Prueba o ejecución del programa.

Para hacer la prueba escribimos en paréntesis el nombre de la función con los valores dados a los datos de entrada

```
;Prueba

(suma 7 8)

(suma 20 30)
```

Aquí hicimos dos pruebas, los valores devueltos deben coincidir con los del ejemplo

Observe que esta instrucción no va antecedida por ; (punto y coma), ya que se necesita que sea leída por el computador

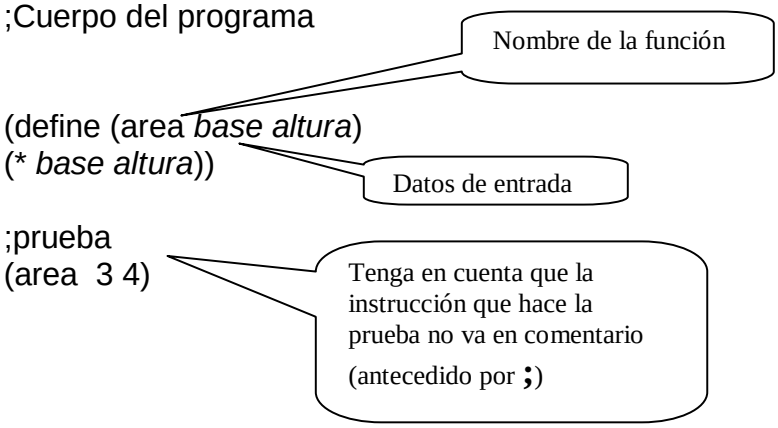
***Importante*** *Si no se hace la prueba el computador no va a realizar el programa porque es aquí donde le decimos a la maquina que ejecute el programa.*

**Ejemplo**
Hacer un programa para hallar el área de un triangulo

;Propósito: calcular el área de un triángulo

;Analisis
;nombre función:AreaTriangulo
;Datos entrada:area: number  y base:number
Tipo de dato de salida: number

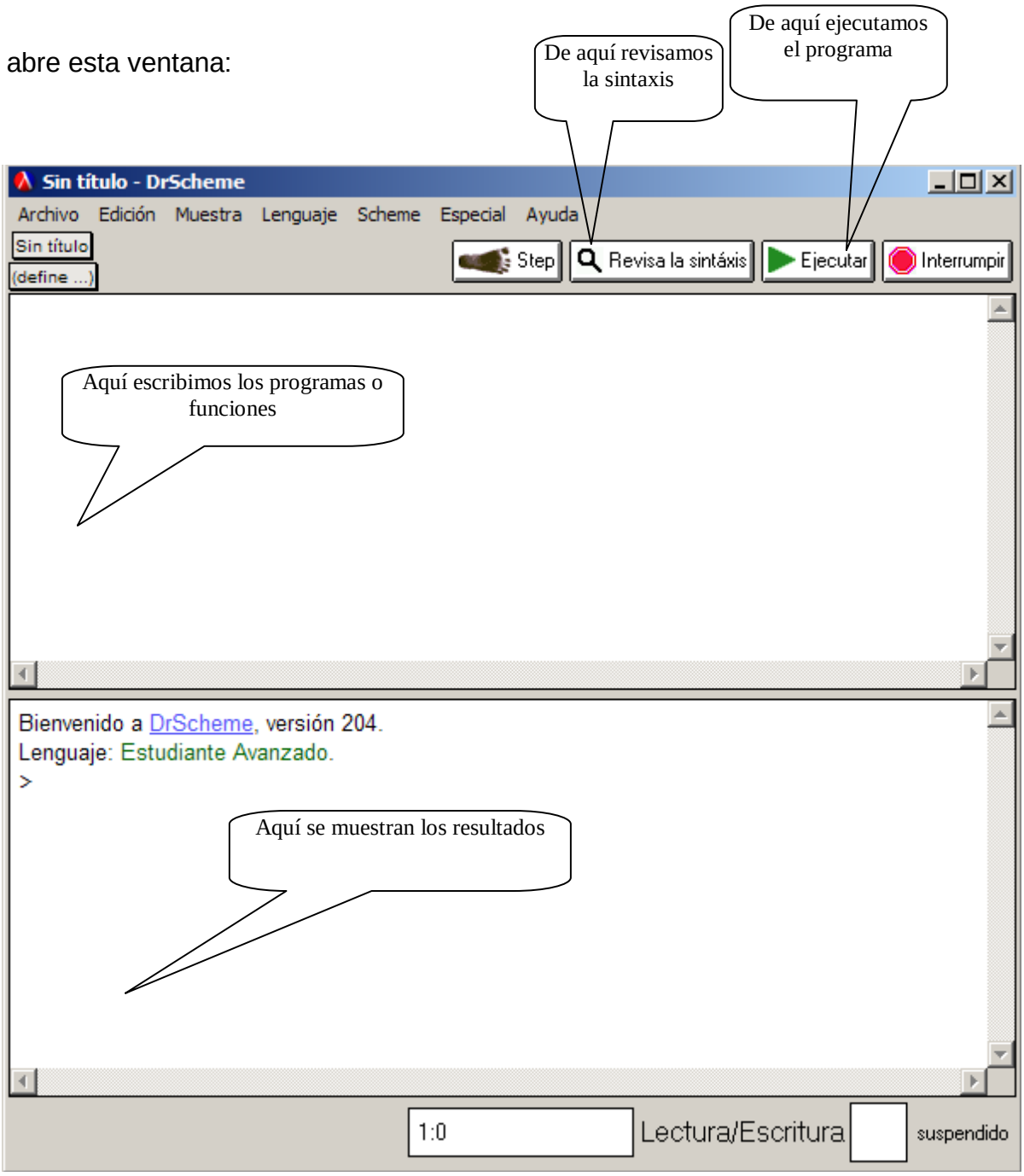
;Cuerpo del programa



## Laboratorio 1 Trabajando en el entorno de Dr Scheme

1) Hacemos clic en
Inicio->Todos los programas->PLT Scheme->Dr Scheme

Se abre esta ventana:



**Este es el entorno de Scheme**

Donde aparecen:

**La Ventana de definicione**s: corresponde a la parte superior de la ventana, allí se escriben las funciones, teniendo en cuenta todos los pasos anteriores.

**Ventana de interacciones**: Corresponde a la parte inferior, y aquí van a aparecer los resultados de la ejecución de las funciones o evaluación de las expresiones.

**En la ventana de definiciones digitamos el siguiente código**

```
;proposito: Realizar la suma de dos números
;análisis
;nombre de la función :suma
      ;datos de entradas: x:y:number
      ;tipo de dato de salida number

;cuerpo del programa
      (define  (suma x  y)
(+ x  y))

      ;prueba

(suma 45  67)
```

3) Guardemos el programa

Archivo->Salvar definiciones como

4) Ejecutemos el programa

Cuando se ejecuta un programa o se revisa su sintaxis pueden aparecer diferentes errores

**Algunos Errores Típicos**

**a) reference to undefined identifier: referencia a un identificador indefinido**

**Posibles causas**

- Cuando un operador esta unido al operando
- Se ha escrito mal alguna variable

**b) read: expected a ')'; indentation suggests a missing ')'**
**Posibles causas**

d) Falta algún paréntesis por cerrar

**c) cond: expected a clause with a question and answer, but found a clause with only one part**

g) Esta mal escrita la condición.
h) Falta el operador *else* en la clausula.

**5) Continue realizando el siguiente programa**

- **Para crear un nuevo programa:** Archivo->nuevo

```
;Proposito: Determinar si un n dado es solución de la ecuación

;Análisis
;nombre de la función
;datos de entrada: n:number
;tipo del dato de salida:string
;Cuerpo del programa

(define (soluciones n)
(cond
[(= 462 (+(* 4(expt n 2)) (* 6 n) 2)) "Si es solución"]
[else "No es solución"]))

;Pruebas o llamado de la función

(soluciones 10)
(soluciones 12)
(soluciones 14)
```

## TALLER 3. FUNCIONES SIMPLES

1. La presión ,el volumen y la temperatura de una masa de aire se relacionan por la formula: Masa=(presion*volumen)/ (0.37 * (temperatura+460). Realizar una función que permita hallar la masa conociendo la formula anterior.

2. Hacer una función que calcule el nuevo salario de un obrero si se sabe que obtuvo un incremento del 25% sobre su salario anterior.

3. El dueño de una tienda compra un artículo a un precio determinado. Hacer una función para obtener el precio en lo que se debe vender para obtener una ganancia del 30%

4. Todos los lunes, miércoles y viernes, una persona corre la misma ruta y cronometra los tiempos obtenidos. Realizar una función para determinar el tiempo promedio que la persona tarda en recorrer la ruta en estos tres días.

5. Hacer un programa para calcular la distancia entre dos puntos dados por la formula $\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$

6. Elabore un programa que permita calcular el área y el volumen de una esfera. El área está dada por $4\ \pi\ r^2$ y el volumen por $\frac{4}{3}\pi\ r^3$

7. Dado 5 números *a, b, c, d, e*. Diseñe un programa que permita calcular el promedio de ellos.

8. Calcular el área de un círculo en función del radio.

Area = $\pi\ r^2$

9. Calcular el área de un circulo del cual se conoce su diámetro (el diámetro es igual a dos veces el radio)

10. Calcular la distancia recorrida por un cuerpo que
es lanzado verticalmente hacia abajo con una velocidad inicial $V_o$.
$S = V_o\ t + ½\ g\ t^2$ $g = 9.81\ m/s^2$

10. Leer el peso en libras de una persona y devuelva el peso respectivo en kilogramos.

Libra = 0.453592 Kg.

12. Convertir un dato de metros a pie y pulgadas (1 *metro*=39.37 *pulgadas*) (1 *pie*=12 *pulgadas*.)

13. Hacer una función que calcule el área de un cuadrado

14. Una empresa paga $8000 por hora a todos sus empleados, Desarrolle el programa que determine el ingreso del empleado.

***Nota:*** No olvidar hacer la definición de constantes, y/o variables auxiliares en los casos que sea requerido.

**Algunos operadores de comparación**

| **Operador** | **Función** | **Ejemplo** |
|---|---|---|
| string? | Para revisar si una variable o valor es de tipo cadena ( string) | (string? “Marisol”)<br>Retornara true ya que “Marisol” es de tipo cadena o string. |
| string=? | compara si dada dos | (string “Cali” “Bogota”) |

| | | |
|---|---|---|
| | valores de tipo string (llamado también alfanumerico o cadenas) son iguales, en caso de serlo devuelve true en caso contrario retorna false | Devuelve false ya que son cadenas diferentes |
| symbol? | Determina si una variable es de tipo symbol | (symbol? ‘Juguete)<br>Retorna true |
| symbol=? | determina si dos símbolos son iguales en caso afirmativo devuelve true, en caso contrario devuelve false | (symbol=? ‘Barbie ‘Barbie)<br>Como los dos símbolos son iguales devuelve true |
| boolean? | Revisa si una variable o valor es de tipo booleano (también conocido como tipo lógico) | (boolean? (> 5 7))<br>Como (> 5 7) retorna false y false es de tipo booleano entonces toda la expresión retorna true. |
| boolean=? | Revisa si dos datos de tipo symbol son iguales | (boolean=? true false)<br>devuelve false |

**CONDICIONALES EN SCHEME**

Permiten resolver problemas donde existen diferentes alternativas o caminos a tomar en su desarrollo.

Dentro de Scheme existen varios condicionales

**Sentencia if**

Veamos su sintaxis:

**(if (condicion) (accion1) (accion2))**

Se evalúa una condición y si si se cumple que es verdadera se realiza la accion1, en caso contrario, es decir cuando la condición evaluada es falsa entonces se debe realizar la accion2.

**Ejemplo**
Determinar si un número es par o no

Como se sabe un número es par cuando es divisible por 2 es decir el residuo de la división del número por 2 es 0, para lo cual utilizamos el operador *remainder*.

;Proposito: determinar si un numero es par o no

;Análisis

Cuando a condición se cumpla o sea que sea verdadero que el residuo de n con 2 es igual a 0, se mostrara por pantalla el mensaje “Es Par” (acion1) y cuando no se cumpla esta condición es decir sea falsa en tonces se mostrara por pantalla el mensaje “Es Impar” (accion 2)

```
;nombre función:ParImpar
;datos de entrada: n:number
;Tipo dato de salida:number

;Cuerpo del programa

(define (ParImpar n)
  (if ( = remainder n 2) 0)   “Es par”    “Es impar”))

;Prueba
(ParImpar 10) devuelve “Es par”
(ParIMpar 7) ;devuelve “Es impar”
```

**Sentencia cond**

Se evalúan varias condiciones y cuando se encuentra alguna que sea verdadera se realiza la acción que la acompaña.  Cuando se realiza una acción la función termina, es decir las otras condiciones no se evalúan.

Al final de las condiciones puede ir la clausula else que significa que si no se cumplió ninguna de la condiciones anteriores entonces se debe realizar la accionN

Sintaxis

```
(cond
(condicion1   accion1)
(condicion2  accion3)
.
.
.
(condicionN  accionN)

(else   accion3))
```

**Ejemplos usando cond**

Determinar si un *n* dado es solución de la ecuación

```
;Propósito: Determinar si un n dado es solución de la ecuación
;Análisis
;nombre de la función: soluciones
;datos de salida: n:number
;tipo datos de salida: number
;Cuerpo del programa
```

```
(define (soluciones n)
(cond
[(= 462 (+(*  4(expt  n  2)) (*  6  n) 2)) “Si es solución”]
[else “No es solución”]))

;Prueba o llamado de la función

(soluciones 3); Retorna  “No es solución”
```

**Ejemplo**

En una Universidad existen tres programas académicos que son: Licenciatura Matemáticas, Tecnología en Electrónica, y Tecnología de Sistemas, los cuales tienen un costo de 1.000.000 para Tecnología Electrónica, $1.200.000 para Matemáticas y  $1.300.000 para Tecnología de Sistemas, además de esto los estudiantes tienen un descuento del 10% si pertenecen a estrato 2 y del 20% si pertenecen a estrato 1:  Hacer un programa que dado el estrato al cual pertenece el estudiante y la carrera que quiere estudiar determine el costo de su matrícula.

```
;Proposito: Determinar el costo de una matricula

;Análisis
;nombre función: costo
;Datos de entrada: estrato :number    y   carrera: number
;Tipo dato de salida: number

;Definición de variables

(define TE “Tecnología en electronica”)
(define TS “Tecnología de Sistemas”)
(define M  “Matematicas”)

(define costo carrera estrato)
(cond
[(and (string=?  carrera  TE) (=  estrato  1)) (- 1000000  (*  1000000   0.2))]
[ (and (string=?  carrera  TE) (=  estrato  2)) (- 1000000  (*  1000000   0.1))]
[(and (string=?  carrera  TS) (=  estrato  1)) (- 1300000  (*  1300000   0.2))]
[(and (string=?  carrera  TS) (=  estrato  2)) (- 1300000  (*  1300000   0.1))]
[ (and (string=?  carrera  M) (=  estrato  1)) (- 1200000  (*  1200000   0.2))]
[(and (string=?  carrera  M) (=  estrato  2)) (- 1200000  (*  1200000   0.1))]))

;llamado de la función
(costo  TE  2)  ; debe devolver  900.000
(costo  M  1); debe devolver 960.000
(costo “ Matematicas”  1) ;debe devolver 960.000
```

Definir variables es opcional, pero se ha hecho para aprender su utilidad.

Se saca el porcentaje y se descuenta al valor original de la matricula.

El llamado de la función se puede hacer con las variables ya definidas o con el nombre de la carrera.

## TALLER 4 CONDICIONALES EN SCHEME

1. Responda que valores devuelven las siguientes expresiones:
   - (string=? “Hola Mundo” “hola Mundo“)
   - (remainder (/ 10 2) 5)
   - (number? 3 .4)
   - (boolean? (> 6 7))
   - (symbol=? ‘mi_mundo ‘Mi_mundo)

2. Determina si las siguientes funciones poseen errores y corrígelas

a) (*define* (*m p q* )(+ 8 (+(* *p q*) *r*)))
b) (*define h* (+ 8 7))
c) (*define* (*multiplicación m k*)
*(cond*
*[(= 1 k) m]*
*[(> m 1) (+ m multiplicacion m (k- 1))]))*

3. Realizar una función que permita calcular el total que una persona debe pagar en una llantería, si el precio de cada llanta es de $300.000, si se compran menos de 5 llantas; y de $ 200.000 si se compran 5 o más.

4. haga una función que lea un número y escriba el valor absoluto del mismo.

5. Desarrolle una función para determinar si un *n* dado es solución de la ecuación $4n^2 + 6n + 2 = 462$

6. Desarrolle una función llamada soluciones que reciba los coeficientes *a, b, c* de una ecuación cuadrática de la forma $ax^2 + bx + c = 0$ y determine cuantas soluciones tiene la ecuación, suponiendo que *a* no es 0

Recuerde que una ecuación cuadrática tiene:
Dos soluciones si $b^2 > 4ac$
Una solución si $b^2 = 4ac$
No tiene solución si $b^2 < 4ac$

Por ejemplo**:** Si *a*=1, *b*=0 y *c*=– 1 la ecuación tiene 2 soluciones. Si *a*=2, *b*=4, *c*=2 la ecuación tiene 1 solución.

7. En un hospital existen tres áreas; ginecología, pediatría, traumatología. El presupuesto anual del hospital se reparte conforme a la siguiente tabla

| **Área** | **Porcentaje del presupuesto** |
|---|---|
| Ginecología | 35% |

| | |
|---|---|
| Traumatología | 25% |
| Pediatría | 30% |

Obtener la cantidad que recibirá determinada área del hospital para cualquier monto presupuestal.

7. En un supermercado se hace una promoción, mediante la cual el cliente obtiene un descuento dependiendo de un numero que se escoge al azar, si el número escogido es menor que 74 el descuento es el 15% sobre total de la compra, si es mayor o igual a 74 el descuento es el 20%. Haga una función que obtenga cuánto dinero se le descuenta.

8. En una fábrica de computadoras se plantea ofrecer a los clientes un descuento que dependerá del número de computadoras que compre. Si las computadoras son menos de 5 se le dará un 10% de descuento sobre el total de la compra, si el número de computadoras es mayor o igual a 5 pero menos de 10 se le otorgara un descuento del 20%; y si son 10 o más se les da un 40% de descuento. El precio de cada computadora es de $1.100.000. Realice una función que calcule cuánto se paga finalmente por la compra de las computadoras.

10. Haga una función que lea dos numeros los multiplíque y muestre su resultado, solo si el segundo número es negativo.

11. Lea tres números distintos, valide que no sean iguales y determine cuál es el mayor.

12. Elabore un programa que lea un número y determine si es par o impar.

13. Una Aerolínea desea saber cuánto debe cobrar a sus pasajeros por la compra de un tiquete, dependiendo de las siguientes condiciones:
    Los destinos de la Aerolínea son Bogotá y San Andrés
    Un tiquete se puede comprar faltando 2, 1 o el mismo día del viaje (0)
    Un tiquete comprado a Bogotá faltando 2 días tiene un costo de $ 150000, faltando 1 día su costo se eleva en un 5% y el mismo día en un 8% más que faltando 1 día, es decir un 8% más sobre el valor del tiquete faltando 1 día.
    A San Andrés el costo del tiquete es de $ 395.000 si es comprado faltando 1 día, si es faltando 2 días su costo se reduce en un 7% y si es el mismo día el valor es mayor en un 5% sobre su valor.

## FUNCIONES AUXILIARES

El uso de funciones auxiliares hace que el diseño de programas sea más manejable y agradable, permitiendo descomponer un problema en pequeños subproblemas.

Se pueden definir múltiples funciones auxiliares, el orden de definición no afecta su llamado.

Desde una función se puede llamar a otra, por ejemplo si diseñamos una función que calcule el área de un círculo seria de la siguiente forma.

```
;Propósito: función para calcular el área de un circulo

;Definición de la variable
(define pi 3.14)

;Análisis de datos
Nombre de la función:AreaCirculo
Datos de entrada: radio:number
Tipo del dato de salida:number

;Cuerpo de la función
(define (AreaCirculo radio))
 (* pi radio radio))
```

Ahora si luego queremos hacer una función para calcular el volumen de una esfera, pero sabemos que el volumen de una esfera se puede calcular multiplicando el área del círculo por el radio, entonces como ya tenemos diseñada la función que calcula el área del círculo podemos utilizarla para calcular el volumen de la esfera.

Entonces diseñemos la función para calcular el volumen de la esfera, y que utilice la función anterior.

```
;Función Principal
Proposito: calcular el volumen de una esfera

;Analisis de datos
;Nombre de la función:VolumenEsfera
;Datos de entrada: r :number
;Tipo de dato de salida number

;cuerpo de la función
(define (VolumenEsfera r)
          (* ( / 4 3) (AreaCirculo r)))

;llamado de la función o prueba
(VolumenEsfera 5)
```

Aquí llamamos a la función auxiliar

**Ejemplo**
Hallar el área de un anillo usando funciones auxiliares

**;Función auxiliar**

;Propósito: calcular el área de una  circunferencia

;Análisis de datos
;Nombre de la función:area_circunferencia
;datos de entrada: radio:number
;Tipo de datos de salida: number

;Cuerpo de la funcion
(define (area_circunferencia *radio*)
(* 3.14 *radio radio*))

;las pruebas por ahora no las hacemos (la función es ejecutada en la función principal)

**;Función principal**

;Propósito calcular el área de un anillo

;análisis de datos:areaAnillo:number:number->number

;Cuerpo del programa

(define (areaAnillo rexterior rinterior)
(- (area_circunferencia rexterior) (area_circunferencia rinterior)))

;Prueba
(areaAnillo 5 4)

## TALLER 5 FUNCIONES AUXILIARES

1. Reducción:

i) explicar cómo es el proceso de reducción y aplicar las funciones con los siguientes valores : *x*=3, *x*=8, *s*= 3 *y*=5 respectivamente

(define (*f x)* (+ (*g x*) 9))
(define (*g y*) (sqrt (* *y* (* (+ 2 12) (/ (* (+ 3 5) (/ 30 10)) 2)))))

(define (misterio *s y*) (expt (misterio1 *s*) (misterio2 *y*)))
(define (misterio1 *x1*) (*(+ *x1* 5) (expt (sqrt 9) 5)))
(define (misterio 2 *x2*) (+(/ *x2* 5) (* (/ 8 9) (+2 4))))

Utilizando funciones auxiliares desarrolle las siguientes funciones:

2. Desarrolla un programa que calcula el área del cilindro. El programa recibe el radio del disco de la base del cilindro y su altura.

3. Un alumno desea saber cuál será su promedio general en las tres materias más difíciles que cursa. Estas materias se evalúan como se muestra a continuación:

La calificación de matemáticas se obtiene de la siguiente manera:
Examen 90%. Promedio de las tares 10%; En esta materia se pidió un total de tres tareas.

La calificación de física se obtiene de la siguiente manera:
Examen 80%. Promedio de tares 20%; En esta materia se pidió un total de dos tareas.

La calificación de química se obtiene de la siguiente manera:
Examen 85%. Promedio de tareas 15%; En esta materia se pidió un total de tres tareas.

## RECURSIVIDAD EN SCHEME

La Recursión o Recursividad consiste en realizar una definición de un concepto en términos del propio concepto que se está definiendo.

Ejemplos:

- ·Los números naturales se pueden definir de la siguiente forma:

0 es un Número natural y el sucesor de un número natural es también un número natural.

- ·El factorial de un número natural n, es 1 si dicho número es 0,

o n multiplicado por el factorial del número n-1, en caso contrario.

- La n-ésima potencia de un número x, es 1 si n es igual a 0, o el producto de x por la potencia (n-1)-ésima de x, cuando n es mayor que 0.

En todos estos ejemplos se utiliza el concepto definido en la **propia definición.**

La Recursividad Es la forma en la cual se especifica un proceso basado en su propia definición. Siendo un poco más precisos, y para evitar el aparente círculo sin fin en esta definición, las instancias complejas de un proceso se definen en términos de instancias más *simples*, estando las finales más simples definidas de forma explícita.

**Nota**: aunque los términos "recursión" y "recursividad" son ampliamente empleados en el campo de la informática, el término correcto en castellano es recurrencia.

### Escritura de programas recursivos

La recursión permite expresar un problema en términos de sí misma, La especificación de una función recursiva no es simple y podemos identificar dos partes en su formulación, la primera es identificar los casos básicos y la segunda definir el paso recursivo, los caso básicos corresponden a los casos limite. La ejecución de una función recursiva debe converger siempre a los casos básicos especificados por el algoritmo.

### Ejemplo

Realizar una función que sume los *n* primeros números, sumar los n primeros números es igual a la suma de los n-1 términos más n y así sucesivamente hasta llegar a 1.

```
;Propósito: sumar los n primeros números
```

```
;Analisis de datos
;nombre de la función: suma
;datos de entrada:n: number
;tipo del dato de salida :number

;Cuerpo del programa

(define  (suma n)
(cond
[(> n 1) (+ n (suma (- n 1)))]
[else 1]))

;prueba
(suma  5)
```

En una función recursiva en vez de llamar una función auxiliar diferente se llama a ella misma y este proceso se repita hasta que se cumpla una condición de parada

**Ejemplo**
Realizar una función recursiva que permita hallar la suma de la serie de los números impares, desde 1 hasta n.

```
;proposito:sumar los numeros impares

;Analisis de datos
;Nombre de la función :impares
;datos de entrada:n: number
;tipo del dato de salida: number


;cuerpo del programa

(define (impares n)
(cond
[(or (= n 1)  (= n 0))1]
((= (remainder n 2) 0)  (impares (- n  1 )))
 (else (+ n (impares (- n  2)))) ))

;Prueba
(impares 5); devuelve 9
(impares 10) ;devuelve 25
```

Internamente el programa hará algo así, aunque nosotros solo vemos el resultado final. (Prueba paso a paso)

(impares 5)

| n | (impares n) |
|---|---|
| 5 | ( + 5  (impares 3))=(+ 5 4)=9 |
| 3 | ( + 3 (impares 1))=(+ 3 1)=4 |
| 1 | 1 |

(impares 9)

| n | (impares n) |
|---|---|
| 10 | ( impares 9) |
| 9 | ( + 9 (impares 7))=(+ 9 16)=25 |
| 7 | (+ 7 (impares 5))= (+ 7 9)=16 |
| 5 | (+ 5 (impares 3))=(+ 5 4)=9 |
| 3 | (+ 3 (impares 1))=(+ 3 1)=4 |
| 1 | 1 |

**Ejemplo**
Hacer un programa que realice la suma de los números pares.

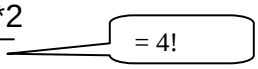
```
;Propósito: sumar los primeros n numero pares

;Análisis de datos
;datos de entrada: n:number
;tipo de datos de salida:Number

;Cuerpo del programa

(define (par n)
(cond
[(or (= n 2)  (= n 0)) n]
[else (+ n (par (- n 2))]

;Prueba
(par 10); devuelve 30
```

Ejercicio: hacer la prueba paso a paso

**Ejemplo**
Calcular el factorial de un número

Sabemos que el factorial de un número son las multiplicaciones del numero con todos los números que hay por debajo del hasta llegar a 2.

5!=5*4*3*2

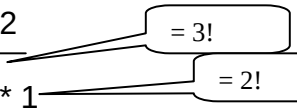



o sea que

5! = 5* !4

y si no damos cuenta

4!= 4* 3 * 2




Y 3!=3* 2 * 1

2!=2* 1

Entonces definamos la función factorial

;Proposito: calcular el factorial de un numero

;análisis de datos
;nombre de la función factorial
;datos de entrada:m:number
;tipo de dato de salida:number

;Cuerpo de la función

```
(define (factorial m)
(cond
((> m 1) (* m (factorial (- m 1))))]
[else 1]))
```

;prueba
(factorial 5) ;Retorna 120

;la ejecución del programa paso a paso es así:

| m | (Factorial m) |
|---|---|
| 5 | ( * 5 (factorial 4))=120 |
| 4 | (* 4 (factorial 3))=24 |
| 3 | (* 3 (factorial 2))=6 |
| 2 | (* 2 (factorial 1))=2 |
| 1 | 1 |

## TALLER 6 RECURSIVIDAD

Hacer funciones recursivas para los siguientes ejercicios:

1. Hallar la suma de los *N* primeros número pares

a. Calcule los múltiplos de 7 hasta 2000

b. Lea un número y determine si es primo o no; un número es primo si no es divisible por otro número diferente de el mismo o la unidad.
Por ejemplo el 7 aparte de el mismo y 1 no hay otro número que lo divida, por lo tanto es primo.

5. Escriba un programa que lea un entero positivo y muestre su equivalente en binario, ejemplo: 1=1, 2=10, 3=11, 4=100.... 20=10100 (investigar)

6. Una función para calcular la serie de fibbonaci

7. Hacer una función para pasa un numero de decimal a binario.

**Nota**: para cada ejercicio mostrar la prueba paso a paso.

## ESTRUCTURAS

Las estructuras son tipos de datos compuestos es decir cuando pensamos en un dato de tipo estructura pensamos en un dato que a su vez tiene otra información como por ejemplo.

Estudiante con los campos o información nombre, cedula, código, curso, edad.

Empleado con los campos de información Salario, nombre, edad, sexo.

Animales con los campos de información Especie , Numero de patas.

Fruta con los campos color, sabor

Película con los campos Género, año, director.

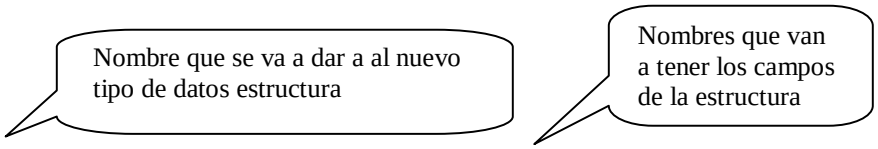
CD Con los campos nombre, número de canciones precio

Amigos con los campos Nombre, teléfono, edad, fecha de cumpleaños

### Definición de una estructura

Como la estructura es un nuevo tipo de datos que entonces tenemos que especificar cómo va a estar conformado, cuáles y de qué tipo van a ser sus campos. Es el programador el que determina como va a estar formada cierta estructura, es decir que campos va a contener la estructura y de qué tipo van a hacer sus campos.

La definición de una estructura se hace de la siguiente forma:

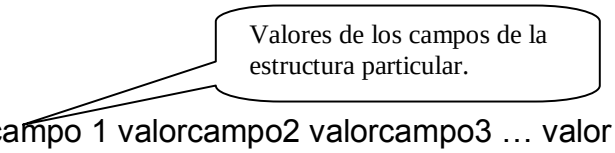



(define-struct nombreEstructura (nombrecampo1 nombrecampo2 nombrecampo3 nombrecampoN)

### Creación de la Estructura particular

Para dar valores a la variable del nuevo tipo de datos hay un operador o constructor que permite hacer esto

Que es make y se usa así:




(make-nombreEstructura (valorcampo 1 valorcampo2 valorcampo3 … valor campoN)

**Selectores**

Son las operaciones que nos van a permitir acceder a los elementos de una estructura

(nombreCampo-NombreEstructura (EstructuraParticular))
ejemplos

**Ejemplo**
Manejemos una estructura CD donde nos interesa el nombre, número de canciones, autor y precio. Como vemos la estructura va a estar formado por otros campos donde los campos nombre, autor son de tipo string y los campos numero de canciones y precio son de tipo numérico. Definamos la estructura

(define-struct CD (nombreCD autor Ncanciones precio))

La anterior es la definición del nuevo tipo de datos como tal.

Ahora pensemos en algunos ejemplos de estructuras CD es decir algunas estructuras en particular.

(make-CD (“Donde Jugaran los NIños” “Mana”10 50000))

**Ejemplo**
Trabajemos ahora con una estructura Fruta

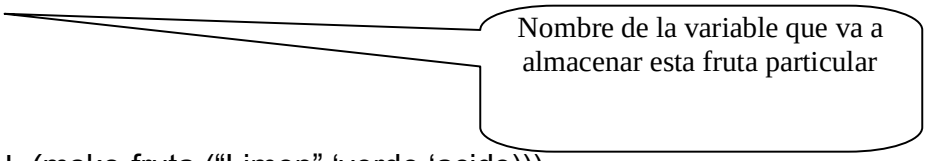
;Definición de la estructura Fruta
(define-struct Fruta (nombre color sabor))

;Definición de algunas frutas particulares

(make-fruta (“Manzana” ‘roja ‘acida))
(make-fruta (“Limon” ‘verde ‘acido))
(make-fruta (“Mango” ‘amarillo ‘dulce))

Para definir la estructuras particulares es conveniente guardar estos valores en variables, para que cuando hagamos referencia a cierta fruta o estructura particular no tengamos que escribir toda la definición sino solamente el nombre de la variable donde fue almacenada. Como se muestra a continuación:

(define M (make-fruta (“Manzana” ‘roja ‘acida)))




(define L (make-fruta (“Limon” ‘verde ‘acido)))
(define mango (make-fruta (“Mango” ‘amarillo ‘dulce)))

***Nota:*** *importante: tenga en cuenta la diferencia entre mango y “mango”, el primero es el nombre de la variable, y el segundo, que aparece entre doble comillas es el valor que se le ha a dado al campo nombre de la estructura particular mango.*

**Aplicación de los selectores**
(Fruta-nombre M) ;retorna “Manzana”
(Fruta-color M) ;retorna ‘roja
(Fruta-sabor M) ;retorna ‘acido

*Ejercicio: aplique los selectores a las frutas L y mango*

Podemos comparar si una variable es de determinado tipo de estructura, colocando el signo **?** a la estructura base seguido de la variable, como se observa a continuación:

(Fruta? M) ; pregunta si M es de tipo Fruta en este caso devuelve true

(CD? L) ; Pregunta si L es de tipo CD, como es falso la expresión devuelve false

## TALLER 7 ESTRUCTURAS

- Defina las siguientes estructuras:

a) Empleado con los campos nombre salario cargo
b) Estudiante con los campos nombre código curso
c) Países con los campos capital extensión población
d) Animal con los campos nombre especie numero de patas

2. Defina tres estructuras particulares de cada uno

3. aplique cada uno de los selectores para obtener los diferentes campos de las estructuras particulares.

## FUNCIONES CON ESTRUCTURAS

Cuando se realizan funciones que manejan estructuras de debe hacer un nuevo paso que es el análisis y definición de datos, donde ira la definición de la estructura. Si no se define la estructura, cuando la utilicemos nos mostrar un error ya que será un tipo de datos desconocido para el programa. También se debe documentar cada campo de que tipo va ser.

También si se quiere se pueden crear algunas estructuras particulares, para utilizarlas en la prueba, pero la prueba se puede realizar con cualquier valor.

Veamos algunos ejemplos con estructuras:

**Hacer una función para calcular el promedio de un estudiante, donde cada estudiante tiene los campos nombre, nota 1, nota 2 y nota 3.**

```
;Análisis y definición de datos
(define-struct Estudiante (nombre nota1 nota2 nota3))
;nombre es de tipo string, nota1, nota2 y nota3 son de tipo number

;Definición de las estructuras particulares

(define estudiante1 (make-Estudiante “Juan Lopez  5 4 3))
(define estudiante2 (make-Estudiante  “Carlos gomez 5 3 2))
(define estudiante3 (make-Estudiante ”Luis salazar” 2.4  3))

;Propósito calcular el promedio de un estudiante

;Nombre de la función: promedio
;dato de entrada: E: Estudiante
;tipo datos de salida: number

; Cuerpo de la función

define (promedio E)
(/ (+ (Estudiante-nota1  E) (Estudiante-nota2  E) (Estudiantes-nota3  E)) 3))

;Prueba
(promedio estudiante1)
(promedio estudiante2)
```

Este es el nombre con el que se definió la estructura La función recibe un dato de tipo Estudiante

**Ejercicio**:Realizar una función que calcule el promedio de tres estudiantes, utilice la función anterior como función auxiliar.

**Ejemplo**
Desarrollar una función que calcule el área de una estructura círculo o rectángulo.

```
;propósito: calcular el área de una estructura círculo o rectángulo.

;Análisis y definición de datos
(define-struct circulo (radio))
;Circulo es una estructura donde su campo radio es de tipo number

(define-struct rectangulo (ancho largo))
;rectángulo es una estructura donde sus campos ancho y largo son de tipo number

;Definición de estructuras particulares
(define circulito (make-circulo 2))
(define rectangulo1 (make-rectangulo 4 5))

;Proposito: Calcular el área de una figura geométrica

;nombre de la función: área
;datos de entrada: figurita:rectanguolo o circulo
;Tipo de datos de salida: number


;Cuerpo del programa

(define (area figurita)
  (cond
    [(rectangulo? figurita) (* (rectangulo-ancho figurita)(rectangulo-largo figurita))]
    [(circulo? figurita) (* (circulo-radio figurita)3.14)]))

;Prueba

(area rectangulo1) ;retorna 20
(area circulito) ;retorna 6.28
```

**Ejemplo**
Hacer una función que reciba una estructura libro y retorne el mismo libro pero con el precio incrementado en 15% si la editorial es oveja Negra, en un 10% si la editorial es Planeta de lo contrario no incremente el precio del libro.

```
;Análisis y definición de datos
(define-struct Libro (nombre autor editorial precio))
;Libro es una estructura donde los campos nombre, autor , editorial son de tipo
string y precio es de tipo number.
```

```
; Definición de estructuras particulares
(define libro1 (make-libro ” 100 años de soledad“  “Gabriel Garcia Marquez “Oveja Negra”  70000))

(define libro2 (make -libro “Mientras llueve  “Fernando Soto Aparicio”  “Planeta” 60000”))

;Proposito: incrementar el precio de un libro

;Nombre de la función: Incrementar

;Dato de entrada L:Libro
Tipo de datos de salida: Libro

;Cuerpo del programa

(define (Incrementa L)
(cond
[(string=? “Oveja Negra” (Libro-editorial L))   (make-libro (libro-nombre  L)  (Libro-autor L)  (Libro-editorial L) (+ (Libro-precio L) (* (Libro-precio L) 0.15)))]
[(string=? “Planeta (libro-editorial L))  (make- libro (Libro-nombre L)  (Libro-autor L) (Libro-editorial L) (+ (Libro-precio L) (* (Libro-precio L)  0.10)))]]
[else L]))

;Esta prueba paso a paso se vería así:

(make-libro “100 años de soledad”  “Gabriel García Márquez”  “Oveja Negra” (+ 70000 (*  70000  0.15)))

;Quedando finalmente
(make-libro “100 años de soledad”  “Gabriel García Márquez”  “Oveja Negra” 80500)
```

Observe que el dato de entrada y salida es el mismo tipo o sea un libro.

Como se modifico un valor de un campo de la estructura entonces la estructura de debe volver a construir.

**Ejemplo**

Desarrolle Un programa que a partir de dos artículos diferentes, donde cada artículo tiene la siguiente información: nombre del artículo,  nombre del almacén donde se vende el artículo, costo de producción unitario, valor venta unitario, número de ventas al año, nombre del almacén  que más ventas al año ha tenido.

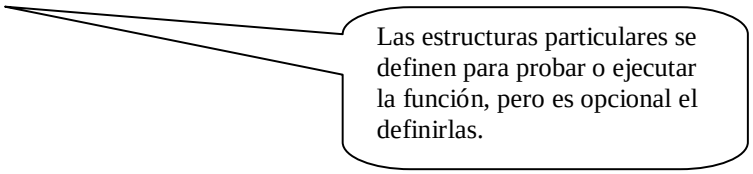
```
;Definición de datos

(define-struct Articulo (nombreArticulo VVUnitario  NVA  Nalmacen CostoProduccion))
```

;Articulo es una estructura donde nombreArticulo Nalmacen son string y CostoProduccion y NVA son de tipo numérico

;Definición de estructuras particulares

(define articulo1 (make-Articulo “Lavadora” 60000 100 “Exito” 50000))

(define articulo2 (make-Articulo “TV” 550000 200 “Carrefour” 400000))




;Propósito determinar el nombre del almacén con el artículo que más ventas al año ha tenido

;Nombre de la función: Ventas
;Datos entrada:A1:Articulo, A2:Articulo
:Tipo de dato de salida: Articulo

;Cuerpo del programa

```
(define (Ventas A1 A2)
(cond
[(> (Articulo-NVA A1) (Articulo-NVA A2)) (Articulo-Nalmacen A1)]
[else (Articulo-Nalmacen A2)])
```

;Prueba
(Ventas articulo1 articulo2); Devuelve “Carrefour”

## TALLER 8 FUNCIONES CON ESTRUCTURAS

1. A partir de las siguientes definiciones y expresiones:

(define *x* 8)

(define *y* 3)

(define-struct nodo (*izq der valor*))

(define-struct estudiante (nombre apellido promedio))

(define z 3)

(define arbolito (make-nodo 10 5 3))

(define chepe (make- estudiante ‘Jose ‘jimenez  4.2))

(define chela (make-estudiante ‘marcela ‘ Hernández  4.3))

(define (misterios x y)
        (+ (+ (* x  x) (* x (* 2  y))) (* y  y))

Seleccione la respuesta adecuada para cada una de las preguntas

El valor de *x* es:
El valor de (misterio *x  y*) es:
El valor de (misterio *y  y*) es:
El valor de (nodo-valor *arbolito*) es
El valor de (estudiante-promedio *chepe*) es:
El valor de (estudiante-apellido *chela*) es:

- Complete las partes señaladas

;Análisis y definición de datos

(define-struct libro (titulo autor editorial precio))

;Propósito: Crear una estructura libro

;Contrato:IngresaLibro: symbol symbol symbol number->libro

,Cuerpo del programa

(define (IngresaLibro tit aud edi pre)

---

2 A partir de dos fechas, cada fecha tiene como información: el día, mes y año, hacer:
   a) Un programa que determine si la primera fecha es anterior a la segunda.
   b)  Un programa que determine cuantos años hay entre las dos fechas

3. A partir de dos estructuras estudiante con campos nombre, curso, nota1, nota2, nota3.  Crear un programa que devuelva el nombre del estudiante con promedio mayor. (Utilice funciones auxiliares)

4. Desarrolle a partir de dos estructuras punto (cada uno con sus coordenadas x y)
   a) Un programa que determine si los puntos son diferentes.
   b) Un programa que determine si están ubicados horizontalmente.
   c) Un programa que determine si están ubicados verticalmente

d) Un programa que retorne la distancia entre los dos puntos.
e) Un programa que retorne el punto más cercano al origen.

5. Desarrolle una función llamada TiempoSegundos, (con los campos horas minutos segundos) la cual recibe una estructura tiempo y genera el número total de segundos. Por ejemplo: si la función recibe como dato de entrada la estructura (make-tiempo 10 10 2) el programa devuelve 36602

6 Defina una estructura de datos que represente un numero fraccionario (que tiene numerador y denominador) y realice una función que múltiple dos números fraccionarios.

7. A partir de una estructura para los datos de estudiante, estos datos son nombre, nota1, nota2, nota3 donde notas son parciales para el estudiante y tienen el mismo valor, desarrollé una función, que retorne la información de si el estudiante aprobó o no la materia.

8. Desarrolle a partir de dos artículos diferentes, donde cada articulo tiene la siguiente información: nombre del artículo, nombre del almacén, costo de producción unitario, valor venta unitario, numero de ventas al año:

a) Un programa que determine si todos los artículos se venden en el mismo almacén
b) Un programa que retorne el articulo que mas ganancia ha dejado en el año.
c) Un programa que retorne el nombre del almacén que menos ventas en total ha tenido
d) Un programa que retorne el valor de venta unitario del articulo que menos se ha vendido en el año
e) Un programa que retorne el promedio total de las ganancias dejadas en el año por los tres .artículos

Aclaración: En un mismo almacén se puede vender más de un artículo.

## LISTAS

Una lista es una secuencia o colección de elementos de igual o diferente tipo, o sea es otro tipo de datos compuesto.

Ejemplos lista de articulos:"jabon", "desinfectante", "papa"

Lista de Universidades "Univalle", "Santiago" "Javerina"

Lista de amigos "Juan" "Pedro" "Pablo"

Lista de ciudades "Cali" "Bogota" "Medellin"

Lista de centros comerciales "Exito" "Carrefour" "chipichape"

Lista de gastos 5000 20000 50000 68000

Lista de utilesEscolares "lapiz" "Borrador" "Cuaderno"

Toda lista con al menos un elemento está conformada por el primer elemento y el resto de la lista.

El resto de la lista son todos los elementos de la lista sin el primer elemento

Para obtener el primer elemento de la lista utilizamos el operador first
Y para obtener el resto de la lista usamos el operador rest

Una lista vacía se representa con la palabra empty

Para escribir una lista en Scheme se usa la palabra list seguida por los elementos de la lista por ejemplo definamos la Lista de centros comerciales "Exito" "Carrefour" "Chipichape"

(list "Carrefour" "Exito" "Chipichape")

También podemos guardar esta lista en una variable

Variable en la que se guarda la lista

(define CentrosComerciales (list "Carrefour" "Exito" "Chipichape"))

### Operadores de listas

| Operador | Función | ejemplo |
|---|---|---|
| first | Obtiene el primer elemento de una | (first (list 'Juan 'Pedro 'Pablo)) ;retorna 'Juan |

| | | |
|---|---|---|
| | lista | |
| rest | Obtiene el resto de una lista | (rest (list ‘Juan ‘Pedro ‘Pablo)) ;retorna (list ‘Pedro ‘Pablo) |
| append | Permite concatenar dos listas | (append (list 5 7 8) (list 10 11)) ; devuelve (list 5 7 8 10 11) |
| reverse | Obtiene la lista al reves | (reverse (list 5 7 8); devuelve (list 8 7 5) |
| cons | Agrega un elemento a una lista | (cons 2 (list 5 7 8) ) ;devuelve (list 2 5 7 8) observe que el elemento que se agrega siempre queda de primero. |

**Problemas que manejan Listas**

Este tipo de funciones son recursivas y por lo general tienen como condición de parada que la lista sea vacía.

En cuanto a la documentación también se debe hacer un análisis y definición de datos para la lista a si como para cualquier tipo de datos compuesto

**Ejemplo**
Realizar una función que calcule el número de elementos de una lista

```
;Análisis y definición de datos
ListaUtiles es una lista de útiles escolares donde útiles escolares es de tipo symbol

;Definición de una lista particular
(define listaUtiles (list ‘sacapuntas ‘Borrador ‘Maletin ))

;propósito función para sumar los elementos de una lista

;Propósito: función que determine el número de elementos de una lista

;Nombre de la función: longitud
,dato entrada:lista de tipo listaUtiles
;tipo dato de salida:number

;Cuerpo del programa
(define (longitud lista))
(cond
[(empty? lista) 0]
[else (+1 (longitud (rest lista)))]))
```

**Ejemplo**

Hacer una función para sumar los elementos de una lista

```
;Analisis y definición datos
listanumeros es una lista de números

;Proposito:Sumar los elementos de una lista

;nombre de la función :sumaElementos
;dato de entrada:L:ListaNumeros
;tipo de dato de salida:number

;Cuerpo del programa
(define (sumaElementos L)
(cond
[(empty? L) 0]
[else (+ (first L)  (sumaElementos  (rest L)))]))

;Prueba
(sumaElementos (list 10  12  5 )) ; devuelve 27
```

**Ejemplo**
Hacer una función que incremente en 100 los elementos de una lista de números.

```
;Análisis y definición de datos

;Proposito: incrementar en 100 cada elemento de una lista  de números.

;ListaNumeros es una lista de de la forma (list a b) donde a es de tipo numero y b
;es el resto de la lista.

;Nombre de la funcion:incremento
;dato entrada:P:ListaNumeros
;tipo dato salida:ListaNumeros

;Cuerpo del programa

(define (incremento P )
(cond
[(empty?  P) emprty]
[(cons (+ 100 (first P)) (incremento (rest P)))] ))

;Prueba
(incremento (list 10 30 50)) ;Devuelve (list 110  130  150)
```

## TALLER 9 LISTAS SIMPLES

1. Desarrollar una función que tome un símbolo, una lista de símbolos y determine sí el símbolo está en la lista.

2. Desarrollar una función que cuente el número de elementos de una lista.

3. Realizar una función que multiplique los elementos de una lista.(utilice las funciones auxiliares de sumar y contar los elementos de una lista)

4. Implementar una función que determine el reverso de una la lista.

5. Desarrollar una función que toma una lista de números y calcula el promedio

6. Crear una función que determine si dos listas son iguales.

7. Desarrollar una función llamada buscar que determine si un numero está en una lista de números.

8 Desarrollar una función que tome un símbolo y una lista de símbolos y determine cuantas veces aparece el símbolo en la lista.

9 Realizar una función que toma un elemento y una lista de números y retorna la lista sin el elemento.

10 Realizar una función que toma una lista de números y dos números (nuevo y viejo) y genera otra lista donde las apariciones del viejo son reemplazadas por las del nuevo.

11. Realizar una función que encuentre todos los divisores de un número y los retorne en una lista.

12. Realizar una función que tome una lista de números y retorne la suma de sus elementos.

13. Hacer una función que tome una lista de números y retorne el producto de todos sus elementos.

## LISTAS DE ESTRUCTURAS

Una de las aplicaciones de lista más comunes es la de permitir el almacenamiento de datos para luego consultarlos. Dicha información podría representar información de una empresa sobre su personal, sueldo, periodo de vacaciones.

Entonces las lista van a estar formadas por estructuras, pero primero se debe definir la estructura para luego si definir la lista que va a estar formada por esas estructuras.

**Ejemplo**
Hacer una función que busque un empleado dentro de una empresa y retorne true (verdadero) si el empleado se encuentra en la empresa y false en caso contrario.

```
; Análisis y definición de datos

(define-.struct empleado  (nombre cedula salario))
;Empledo es una estructura donde nombre es de tipo string, cedula  y salario  de
tipo number.

;Estructuras particulares
(define empleado1 (make-Empleado “Juan Perez”  578910    800 000))
(define empleado2  (make-Empleado “Maria Gomez”  754328  900 000))
(define empleado3 (make-Empleado “Carlos Salazar”  54321  700 000))

;Empresa es una lista formada por estructuras Empleado

;Lista Particular
(define empresa1 (list empleado1  empleado2  empleado3 )

;Proposito:Buscar un empledo en una empresa

;Nombre de la función: buscar
;datos de entrada:n de tipo string  y  E de tipo Empleado

;Cuerpo del programa

(define (buscar n E)
(cond
[(empty E) false)]
[(string=? N  (Empleado-nombre (first E))) true ]
[else  (buscar n (rest E))]))

;Prueba
(buscar “Carlos Salazar” empresa1) ;devuelve true
```

Compara s i *n* y el nombre del primer elemento de la lista son iguales.
Como el primer elemento de la lista es una estructura se debe indicar con cuál de los campos que hay allí se va a hacer la comparación.

**Ejemplo**

Crear una función llamada ElevarPrecios que tome una lista de artículos (el articulo tiene los campos nombre del artículo, precio y código del producto) y produzca la lista con todos los precios de los artículos incrementados en un 5%.

```
;definición de datos
(define-struct articulo (nombre precio código))

;definición de datos particulares
(define L1 (list (make-articulo "medias" 3500 34) (make-articulo "zapatos" 65000
23)))

;cuerpo de la función
(define (incrementar L)
  (cond
    [(empty? L) empty]
    [else (cons (make-articulo (articulo-nombre (first L)) (* (articulo-precio (first L))
1.05) (articulo-codigo (first L))) (incrementar (rest L)))]))

;prueba
(incrementar L1)
```

Cuando algun campo de la estructura de la lista cambia se debe volver a construir la lista insertando de nuevo cada elemento, y cada elemento a su vez se crea con el operador make

## TALLER 10 LISTAS DE ESTRUCTURAS

1. Desarrollar una lista de artículos donde la estructura articulo tenga los campos nombre del artículo, código y precio. Y realizar las siguientes funciones:

   a) Una función que determine el nombre del artículo a partir del precio y la tienda
   b) Una función que determine el precio del artículo a partir del nombre y la tienda

2. Dada una lista de estudiantes de la universidad con los campos nombre, código y curso, realizar una función que dado un código determine si el estudiante se encuentra matriculado en la universidad.

3. Dada un directorio telefónico de amigos donde cada amigo tiene nombre, dirección y teléfono, crear una función que dado el nombre devuelva el número telefónico del amigo.

4. Crear una función llamada ElevarPrecios que tome una lista de artículos (el articulo tiene los campos nombre del artículo, precio y código del producto) y produzca la lista con todos los precios de los artículos incrementados en un 5%.

5. Implementar un diccionario, donde dada una palabra la función devuelva el significado de la palabra.

6. Desarrollar a partir de una estructura CD con los campos nombre del álbum, autor, precio, número de canciones, un programa que reciba la lista de CDS y devuelva el CD de menor precio.

7. El manejo de estructuras dentro de lista permite representar situaciones muy comunes, por ejemplo, los admitidos a un plan de estudios, ya que se podría crear una lista cuyos elementos son la información de cada aspirante seleccionado.

8. En los siguientes problemas se debe considerar que un curso tiene como información: el nombre de la asignatura (materia), grupo, créditos, salón, profesor hora de inicio, hora finalización. Además, debe considerar que una escuela, por ejemplo, la escuela de ingeniería de sistemas es la encargada de ofrecer determinados cursos dentro de la universidad. Es decir una unidad académica se podría modelar considerando que tiene un nombre y una lista de cursos de cursos que son ofrecidos. Con base en lo anterior.

   a) Desarrolle un programa que a partir de una escuela, determine cuál es el total de créditos que ofrece.

b). Desarrollar un programa que a partir de una escuela, y del nombre de una asignatura, determine cuantos cursos diferentes ofrece la escuela de esa asignatura.

c) Desarrollar un programa que a partir de una escuela y del nombre de un profesor, retorne la información de los cursos que el enseña.

9 Desarrollar un programa que a partir de una escuela determine cuantos profesores diferentes enseñan en ella (considere que un profesor puede enseñar más de un curso).

## MEMORIA Y FUNCIONES

La memoria en Scheme se usa en programas que requieren tener en cuenta ejecuciones pasadas. Para manejo de memoria se utiliza la instrucción set! que permite modificar una variable ya definida antes, si la variable no se ha definido antes no se puede modificar.

Recordemos que una variable s define así:

(define variable expresion)

Para modificar el valor de variable lo hacemos asi:

(set! variable expresión)

## SECUENCIACION

En Scheme se puede establecer órdenes de las instrucciones utilizando begin de la siguiente forma:

(begin (expresión1)
(expresion2)
(expresion3)
(expresión)
(expresión N + 1))

Todas las expresiones se evalúan desde la primera hasta la N luego se evalúa la expresión N + 1 y su valor es devuelto por toda la expresión begin N

**Ejemplo**

Para definir una variable se usa la palabra ***define*** y solo se puede hacer una vez

(define x 10)

Si luego queremos modificar el valor de la variable *x* se debe hacer a través del operador ***set!***
(set! x (+ x 12)) ; se le suma 12 a x, quedando valiendo 22
(set! x (* x 2)) ; se multiplica x por 2, entonces queda valiendo 44
X ; se imprime 44

Veamos otro ejemplo con una lista de números
(define l1 (list 2 3 4 ))
(set! l1 (cons 5 l1)) ; se le inserta el elemento 5 a la lista
"al insertar 5 l1 queda"
(list 5 2 3 4)

Veamos ahora la utilización de memoria dentro de las funciones
Primero hagamos una función que agregar elementos a la lista pero sin utilizar memoria

```
(define (agregarlista L e)
  (cons e L))

"al llamar la funcion primera vez"
(agregarlista l1 6)
, la función retorna la lista"
```

(list 6 5 2 3 4)

"pero si se imprime la variable ya no recuerda lo que se le inserto, siempre aparece como fue definida en el define"

l1
es decir (list 5 2 3 4)
"si vuelve y se llama a la función, para inseratar 7"
(agregarlista l1 7)

"pero si se imprime la lista fuera de la función vemos que sigue estando como se definió"
(list 5 2 3 4)

```
"ahora con memoria"
(define (agregarlistaconmemoria L e)
  (set! l1 (cons e L)))
```

Se llama la función

```
(agregarlistaconmemoria l1 6)
```

si se imprime l1 queda (list 6 5 2 3 4

```
"y si ahora se inserta el 7"
(agregarlistaconmemoria l1 7)
```

" al imprimirse la lista conserva memoria es decir aparecen todos los elementos insertados" (list 7 6 5 2 3 4)

Veamos un ejemplo de memoria con listas de estructuras

```
(define-struct  amigo (nombre telefono)) ;definimos la estructura
(define directorio empty) ; se define la variable que va a manejar la memoria
```

Ahora agreguemos un elemento al directorio

```
(set! directorio ( cons (make-amigo "pedro" 3164018769) directorio))
"el directorio queda"
(list (make-amigo "pedro" 3164018769))
```

Si se agregar otro elemento

```
(set! directorio (cons (make-amigo "pilar" 3012576845) directorio))
```

el directorio con dos elementos queda

```
(list (make-amigo "pilar" 3012576845) (make-amigo "pedro" 3164018769))
```

"ahora hagamos una función que se encargue de agregar los nombre y teléfonos de nuestros amigos a al directorio del celular"

```
(define (agregarDirectorio nom tel)
(set! directorio (cons (make-amigo nom tel) directorio)))

"llamemos la función"
(agregarDirectorio "juanca" 301234566)
```

Si se imprime la variable directorio esta queda

```
(list (make-amigo "juanca" 301234566) (make-amigo "pilar" 3012576845) (make-amigo "pedro" 3164018769))
```

**Ejemplo**

Haga un programa (con todos los pasos) que maneje un directorio telefónico de amigos, permitiendo las siguientes opciones.

a) Adicionar un registro telefónico al directorio
b) Eliminar un amigo de un directorio dado su nombre.
c) Recuperar un teléfono a partir del nombre presente en el directorio

Solución a)

```
;proposito: agregar un nombre y un teléfono al directorio
```

```
;(define directorio empty)
;análisis
;nombre función: agregar
;Dato entrada :n:string       t:number
;Tipo dato salida: lista de amigos

;cuerpo de la función
(define (agregar n t)
  (begin (set! directorio (cons (make-amigo n t) directorio) ) directorio))

;prueba
(agregar "Ricardo" 5589671);   ; devuelve  (list (make-amigo "Ricardo" 5589671))

Solución b)
;Proposito: eliminar un amigo de un directorio,  con manejo de memoria

;definición de datos

;definición de la estructura
(define-struct amigo (nombre telefono))

;definicion de la lista
(define directorio (list (make-amigo "Carlos" 3324567)
                         (make-amigo "Jairo" 5589645)))

;analisis
;nombre funcion: eliminar
;datos entrada name:string
; L:lista amigos

;cuerpo función
 (define (eliminar name L)
   (cond
    [(empty? L) "no se encuentra en el directorio"]
    [(string=? name (amigo-nombre (first L))) (begin (set! directorio (rest L))
directorio)]
   [else (set! directorio (cons (first L) (eliminar name (rest L))))]))


;prueba
 (eliminar "Carlos" directorio)


; devuelve  (list (make-registro "Jairo" 5589645))

  (eliminar "Jairo" directorio) ;devuelve empty
   (eliminar "Jose" directorio)
```

**Ejemplo**

Hacer una función para determinar el número menor de una lista de números

```
;proposito función para determinar el menor de una lista de números
(define M 0)
(define l1 (list 5 10 2 4 2))

;Analisis: nombre funcion:Menorlista
;dato entrada:L:lista numeros,      M:  number

;cuerpo de la funcion
(define (Menorlista L m)

  (cond
    [(empty? L ) m]
    [(< m (first L)) (Menorlista  (rest L) m)]
    [(> m (first L)) (Menorlista (rest L) (first L))]
   [(= m (first L)) (Menorlista (rest L) m)]))


(define (primero L )
  (begin (set! M (first L)) (Menorlista (rest L) M )))


(primero l1)
```

## TALLER 11 MEMORIA

1. Desarrollar una función para manejar los estudiantes de la Universidad
El programa debe permitir:
a) Agregar estudiantes
b) Consultar por el código todos los datos de un estudiante
c) Borrar estudiantes.

El estudiante tiene los campos nombre, código, curso

2. Crear un programa para el manejo de los empleados de una empresa. El programa debe permitir:
a) Agregar empleados
b) Consultar por la cedula todos los datos de un empleado
c) Borrar empleados de la empresa
d) El empleado tiene los campos nombre, cedula, salario.

3. Crear un programa para el manejo de una agenda electrónica. El programa debe permitir:
a) Agregar registros a la agenda
b) Consultar por nombre el teléfono de una persona
c) Borrar registros de la agenda

## LOCAL

Permite encapsular o ocultar información a las demás funciones.
Utilidad

Sintaxis
<exp>= (local (<def-1> <def_n> <exp>)

```
(local ((define x 4)
(define (suma x y) (+ x y))))
(suma 3 x)
```

**Ejemplo**
El siguiente ejercicio se hace de forma global

```
(define x 5)
(define (suma x y)  (+ x y))
(suma 3 x)  ;Retorna 8
```

Este mismo ejercicio de forma local

```
(local ((define x 5) (define (suma x y)
(+ x y))) (suma 3 x))  ;Retorna 8

(suma 5 x)
;este llamado esta fuera del local por lo tanto no se reconoce y el programa genera un error.
```

**Ejemplo**

Colocar en el cuerpo de local, una expresión que sea un llamado a la función principal que recibe como argumentos los de la función que contiene el local.

Una función que calcula el área de un anillo usando el local.

```
;Contrato:
;areaAnillo: number:number->number

;Proposito
;Un anillo con radio externo 2 y radio interno 1 tiene area 9.42
;Un anilo con radio externo 4 e interno 1 tiene area 2.51

;Cuerpo
(define (areAnillo re ri)
(- (area_circulo re) (area_cierculo ri)))


(define (area_anillo_local rext rint)
(local (
(define pi 3.14
(define (area_circulo r) (* pi 3.14))
(define (are_anillo re ri)
(- (area_circulo re) (area_circulo r1)))
)
(area_anillo rext rint)
)
)

;prueba
(area_anillo_local 2 1) produce 9.42
(area_anillo_local 9.1) produce 25.12
```

## TALLER 12  LOCAL

1. Responda que valor toma *x* en la siguiente expresión

   (*define x (local m ((define (operaciones x y*) (+ (* *x  y*) 6 ))) (*operaciones* 10 12))

   h)  que valor devuelve la siguiente expresión

   (*local (define x* 5) (*define* (*suma x y*) (/ (* (+ *x* 5) 10) 2))) (*suma* 4 x)

3. Que valores devuelven las siguientes expresiones

   a) (local ((define x 3)  (define y 4)
   (define (suma x y z)
   (+ x y z)) (suma x y 10))

   b) (local (( define pi 3.14)
   (define (volumen r) (* (/ 4 3) pi r r)))
   (volumen 5))

   c) (volumen 6)

   d) realizar una función para hallar el promedio de una lista usando local

## Bibliografía


- *Libro How to Designs Programas*
- *http://www.wikipedia.com*