UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA

DEPARTAMENTO DE
INFORMÁTICA Y ANÁLISIS NUMÉRICO

PROGRAMACIÓN DECLARATIVA

INGENIERÍA INFORMÁTICA

CUARTO CURSO

PRIMER CUATRIMESTRE

Tema 2.- Expresiones y funciones

Primera parte: Scheme

- Tema 1.- Introducción al Lenguaje Scheme
- Tema 2.- Expresiones y Funciones
- Tema 3.- Predicados y sentencias condicionales
- Tema 4.- Iteración y Recursión
- Tema 5.- Tipos de Datos Compuestos
- Tema 6.- Abstracción de Datos
- Tema 7.- Lectura y Escritura

Segunda parte: Prolog

- Tema 8.- Introducción al Lenguaje Prolog
- Tema 9.- Elementos Básicos de Prolog
- Tema 10.- Listas
- Tema 11.- Reevaluación y el “corte”
- Tema 12.- Entrada y Salida

Primera parte: Scheme

Tema 1.- Introducción al Lenguaje Scheme

Tema 2.- Expresiones y Funciones

Tema 3.- Predicados y sentencias condicionales

Tema 4.- Iteración y Recursión

Tema 5.- Tipos de Datos Compuestos

Tema 6.- Abstracción de Datos

Tema 7.- Lectura y Escritura

Índice

Índice

1. Elementos básicos de Scheme

- Caracteres
- Cadenas de caracteres
- Números
- Identificadores

1. Elementos básicos de Scheme

- Caracteres
- Cadenas de caracteres
- Números
- Identificadores

1. Elementos básicos de Scheme

- Caracteres
  - Letras
  - Dígitos
  - Caracteres alfabéticos extendidos
  - Espacios en blanco
  - Punto y coma
  - Caracteres especiales

1. Elementos básicos de Scheme

- Caracteres
  - Letras
    - ***a*, *b*, ..., *z*, *A*, *B*, ..., *Z***
    - Las letras **mayúsculas** y **minúsculas** son consideradas **iguales**
    - Excepciones
      - ❑ Forman parte de una cadena: "Cadena"
      - ❑ Forman parte de un literal : 'Literal
      - ❑ Están precedidos de una barra inclinada: \A

1. Elementos básicos de Scheme

- Caracteres
  - Dígitos
    - 0, 1, 2, …, 9

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres alfabéticos extendidos
    - + - . * / < = > ! ? : $ _ & ~ ^
    - Algunos poseen un significado especial
    - Pueden formar parte de un identificador
      - ❑ Fin-de-mes, estrella*, saldo.final, etc.
      - ❑ Convenios de notación
        - ✓ !: se asocia a una función que modifica una variable: editar!
        - ✓ ?: se asocia a un identificador que es un predicado: persona?

1. Elementos básicos de Scheme

- Caracteres
  - Espacios en blanco
    - Espacios en blanco, tabuladores, saltos de línea y saltos de página.
    - Se utilizan para
      - ❑ mejorar la legibilidad del programa
      - ❑ separar los “componentes léxicos”
    - Son significativos cuando forman parte de una cadena de caracteres:
      - ❑ “Cadena con espacios”

1. Elementos básicos de Scheme

- Caracteres
  - Punto y coma: “;”
    - Se utiliza para crear comentarios de una línea

      ; Ejemplo de comentario de una línea

      ;; Más realce: dos puntos y comas

    - Si aparece en una cadena de caracteres, **pierde** su significado especial

      “Punto y coma → ; que no inicia un comentario”

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - Punto: .
    - Más y menos: + -
    - Paréntesis: ( )
    - Comilla simple: '
    - *Backquote*: `
    - Coma: ,
    - Coma y arroba: ,@
    - Comillas dobles: " "
    - Barra: \
    - Corchetes y llaves: [ ] { }
    - Sostenido: #

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - Punto: .
      - ❑ Forma parte de
        - ✓ Identificadores: saldo.final
        - ✓ Números: 2.5
        - ✓ Pares : (a . b)
      - ❑ Indica el comienzo de parámetros opcionales

        (***lambda*** (x1 x2 x3 . z))

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - + -

❑ Operadores prefijos de adición y sustracción

✓ (+ 3 4)

✓ (- 10 7)

❑ Pueden formar parte de

✓ identificadores: fin-de-mes, iva+irpf

❖ Observación: no debe haber espacios

✓ números: -32

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - Paréntesis: *( )*
      - ❑ Delimitadores de
        - ✓ Expresiones aritméticas: (* 2 (+ a 1))
        - ✓ Listas: (a b c d)

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - Comilla simple: '
      - ❑ Permite definir un literal
        - ✓ *'dato*

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - *Backquote*: `
      - Usado para indicar datos "casi" constantes.
      - Permite construir una **lista** o **vector** en los que no se conocen todavía todos sus elementos
      - Este carácter es equivalente al comando *quasiquote y se usa junto con "coma y coma y arroba"*: , ,@

        (continúa) →

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - *Backquote*: `

(define c 3)

`(a b ,c)

➔ *(a b 3)*

(define c 3)

(quasiquote (a b ,c))

➔ *(a b 3)*

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - Coma: ,
    - Coma y arroba: , @
      - ❑ Se utilizan junto con *backquote* para evaluar datos “casi” constantes
        - ✓ , → evalúa una expresión
        - ✓ ,@ → evalúa una lista

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - Coma: ,
    - Coma y arroba: ,@
      - `#(10 5 ,(sqrt 4) ,@(map sqrt '(16 9)) 8)
        - ➔ #(10 5 2 4 3 8)

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - Comillas dobles: " "
      - ❑ Delimita cadenas de caracteres
        - ✓ "Cadena maravillosa"

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - Barra inclinada: \
      - ❑ Permite indicar constantes de tipo carácter
        - ✓ \a
      - ❑ También se usa como carácter de escape dentro de una cadena de caracteres
        - ✓ " Mi \"gran\" amigo"

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - *Corchetes y llaves*: *[ ] { }*
      - ❑ Caracteres reservados para futuras **ampliaciones** del lenguaje.

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - *Sostenido (almohadilla) (1/3): #*
      - ❑ *Constantes lógicas:*
        - ✓ *#t: verdadero*
        - ✓ *#f: falso*
      - ❑ *Carácter constante:*
        - ✓ *#\a*
      - ❑ *Inicio de un vector:*
        - ✓ *#(1 1 1)*

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - *Sostenido (almohadilla) (2/3): #*
      - ❑ *Indica que el número es exacto: #e*
        - ✓ > *#e1e2*
        - ➔ *100*
      - ❑ *Indica que el número es inexacto: #i*
        - ✓ > *#i1e2*
        - ➔ *100.0*

1. Elementos básicos de Scheme

- Caracteres
  - Caracteres especiales
    - *Sostenido (almohadilla) (3/3) : #*
      - ❑ *Número binario: #b101*
      - ❑ *Número octal: #o323*
      - ❑ Número decimal: *#d910*
      - ❑ Número hexadecimal: *#fff*

1. Elementos básicos de Scheme

- Caracteres
- Cadenas de caracteres
- Números
- Identificadores

1. Elementos básicos de Scheme

- Cadenas de caracteres
  - Secuencias de caracteres delimitadas por comillas (").
  - Las comillas pueden formar parte de una cadena de caracteres si van precedidas de la barra inclinada (\).
  - Ejemplos
    - "esto es una cadena"
    - "dato 3 "
    - "Querida \"Marta\": "
    - "100"
    - "==>"

1. Elementos básicos de Scheme

- Caracteres
- Cadenas de caracteres
- Números
- Identificadores

1. Elementos básicos de Scheme

- Números
  - Enteros
  - Racionales
  - Reales
  - Complejos

1. Elementos básicos de Scheme

- Números
  - Enteros:
    - 4, -56, 209,...
    - binarios: #b101 (5 en base 10)
    - octales: #o101 (65 en base 10)
    - decimales: 101 ó #d101
    - hexadecimales: #x101 (257 en base 10)
      #f ó #F (15 en base 10)

1. Elementos básicos de Scheme

- Números
  - Racionales
    - 5/3, -9/7, 37/17,...
  - Reales
    - 34.7, -93.001, 3.0, …
  - Complejos
    - 2+3i, 7.5-3.4i, +i, …

1. Elementos básicos de Scheme

- Caracteres
- Cadenas de caracteres
- Números
- Identificadores

1. Elementos básicos de Scheme

- Identificadores
  - Secuencia de letras, dígitos y caracteres "alfabéticos extendidos" que **no** comienza por un número.
  - Ejemplos:
    - *nuevo*
    - *lista->vector*
    - *X2*
    - *grande?*
    - *cambiar?*
    - *fin-de-mes*

1. Elementos básicos de Scheme

- Identificadores
  - Secuencia de letras, dígitos y caracteres “alfabéticos extendidos” que **no** comienza por un número.
  - Ejemplos: identificadores válidos pero **no** recomendables
    - +
    - **
    - <=?
    - …

1. Elementos básicos de Scheme

- Identificadores
  - **Se debe comprobar** si el intérprete **distingue** las letras mayúsculas y minúsculas
  - Algunos intérpretes de Scheme consideran como iguales a los siguientes identificadores
    - *nuevo*
    - *NUEVO*
    - *Nuevo*
    - *NueVo*
    - *NuEvO*

1. Elementos básicos de Scheme

- Identificadores
  - Los identificadores son utilizados para representar
    - Palabras claves
    - Variables
    - Funciones
    - Símbolos

1. Elementos básicos de Scheme

- Identificadores
  - **Palabras claves**

=>	*and*	*begin*	*case*
cond	*define*	*delay*	*do*
else	*if*	*lambda*	*let*
letrec	*let**	*not*	*or*
quasiquote		*quote*	*set!*
unquote		*unquote-splicing*	

1. Elementos básicos de Scheme

- Identificadores
  - **Variables**
    - Uso de palabras claves como variables
      - ❑ Algunos intérpretes permiten que las palabras claves sean usadas como variables
      - ❑ No es recomendable:
        - ✓ Limita la portabilidad
        - ✓ Aumenta la ambigüedad
        - ✓ Ejemplo: (***define*** define 9)

1. Elementos básicos de Scheme

- Identificadores
  - **Símbolo**
    - Representar un símbolo cuando aparece como un literal o dentro de un literal.
      - ‘uno
      - (*quote* uno)

Índice

2. Expresiones

- Descripción
- Tipos de expresiones

2. Expresiones

- Descripción
- Tipos de expresiones

2. Expresiones

- Descripción
  - Una expresión es una instrucción que devuelve un valor.
  - Las expresiones son escritas mediante una notación **prefija** delimitada por paréntesis.
  - Un programa escrito en Scheme es una secuencia de
    - expresiones
    - y definiciones de funciones y variables.

2. Expresiones

- Descripción
- Tipos de expresiones

2. Expresiones

- Tipos de expresiones
  - Literales
  - Variables
  - Operadores, funciones o procedimientos
  - Formas especiales

2. Expresiones

- Tipos de expresiones
  - **Literales**
  - Variables
  - Operadores, funciones o procedimientos
  - Formas especiales

2. Expresiones

- Tipos de expresiones
  - **Literales**
    - Los literales son expresiones que se autoevalúan
      - ❑ Cada expresión se evalúa a un dato cuya representación gráfica es equivalente a la expresión.

2. Expresiones

- Tipos de expresiones
  - **Literales**
    - Tipos de literales
      - ❑ Números
      - ❑ Cadenas
      - ❑ Constantes lógicas (*#t* y *#f*)
      - ❑ Definidos usando la comilla o *quote*

2. Expresiones

- Tipos de expresiones
  - **Literales**
    - Ejemplos (1/4)

Literal	valor
"muestra"	*"muestra"*
*(**quote** muestra)*	*muestra*
'muestra	*muestra*

2. Expresiones

- Tipos de expresiones
  - **Literales**
    - Ejemplos (2/4)

Literal	valor
3	*3*
*(**quote** 3)*	*3*
'3	*3*
20.5	*20.5*
*(**quote** 20.5)*	*20.5*
'20.5	*20.5*

2. Expresiones

- Tipos de expresiones
  - **Literales**
    - Ejemplos (3/4)

Literal	valor
*(**quote** (a b c))*	*(a b c)*
'(a b c)	*(a b c)*
*(**quote** ())*	*()*
'()	*()*

2. Expresiones

- Tipos de expresiones
  - **Literales**
    - Ejemplos (4/4)

Literal	valor
*(**quote** "hola")*	*"hola"*
"hola"	*"hola"*
#t	*#t*
*(**quote** #t)*	*#t*
'#t	*#t*

2. Expresiones

- Tipos de expresiones
  - Literales
  - **Variables**
  - Operadores, funciones o procedimientos
  - Formas especiales

2. Expresiones

- Tipos de expresiones
  - **Variable**
    - **Identificador** que tiene asociada una **posición de memoria** en la que se puede almacenar un valor.

2. Expresiones

- Tipos de expresiones
  - **Variable**
    - Declaración o definición

*(**define** <nombre> <expresión>)*

- ❑ <*nombre*> es un identificador
- ❑ <*expresión*> es una expresión de Scheme

2. Expresiones

- Tipos de expresiones
  - **Variable**
    - **define**
      - ❑ Reserva una **posición de memoria** que se asocia a la variable.
      - ❑ Si la variable ya existe entonces sólo se produce la **asignación** del valor.

2. Expresiones

- Tipos de expresiones
  - **Variable**
    - Se usan las reglas de ámbito léxico o estático para determinar el valor de una variable.
    - El valor dependerá del contexto más próximo que contenga a la variable.

2. Expresiones

- Tipos de expresiones
  - **Variable**
    - Ejemplos:

*(**define** x 10)*
x ➔ *10*

*(**define** y 20)*
y ➔ *20*

2. Expresiones

- Tipos de expresiones
  - **Variable**
    - Modificación
      - ❑ Definiendo de nuevo la variable

*(**define** a 12)*
a ➔ *12*
*(**define** a 15)*
a ➔ *15*

      - ❑ Uso de la forma especial **set!**

*(**define** a 12)*
(set! a 15)
a ➔ *15*

2. Expresiones

- Tipos de expresiones
  - **Variable**
    - set!
      - ❑ El símbolo de admiración "!" indica que la forma especial **modifica** el valor de la variable
      - ❑ No se puede utilizar la forma especial **set!** sobre una variable que **no** haya sido **previamente** definida o declarada.
      - ❑ Observación:
        - ✓ la **programación funcional** se caracteriza por **evitar** el uso de set!

2. Expresiones

- Tipos de expresiones
  - **Variable**
    - **Scheme**: un lenguaje débilmente tipificado o dinámicamente tipificado
      - ❑ Los tipos **no** están asociados a las variables.
      - ❑ Los **tipos** están asociados a los **valores.**
      - ❑ Una variable puede ser **definida** con un valor de un cierto tipo y **después** se le puede asignar el valor de otro tipo.

2. Expresiones

- Tipos de expresiones
  - **Variable**
    - **Scheme**: un lenguaje débilmente tipificado o dinámicamente tipificado
      - ❑ Ejemplo:

*(**define** clave "luna-roja")*
clave ➔ *"luna-roja"*

*(**define** clave 10)*
clave ➔ *10*

*(**set!** clave 'x)*
clave ➔ *x*

2. Expresiones

- Tipos de expresiones
  - Literales
  - Variables
  - **Operadores, funciones o procedimientos**
  - Formas especiales

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - Secuencia de una o más expresiones encerradas entre paréntesis.

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Sintaxis**

      *(<**operador**> <operando$_1$> ...)*

      ❑ Se utiliza la **notación prefija**

      ❑ Operador:

      ✓ Predefinido: +, *, *sqrt*, *expt*, *acos*, ...

      ✓ Definido por el programador

      ❑ Operando: expresión más simple

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Evaluación**

1. Se evalúan el operador y los operandos
   a. Se usa un orden no especificado
   b. Se utilizan las reglas de ámbito **estático**
   c. Se obtienen un procedimiento y unos argumentos.

2. Se aplica el procedimiento a los argumentos generando el resultado final de la expresión.

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - Los procedimientos definidos por el programador y los suministrados por el intérprete tienen el **mismo tratamiento**.
    - Se aconseja no redefinir operadores predefinidos por el lenguaje.
      - ❑ Ventajas:
        - ➢ Eficiencia
        - ➢ Portabilidad

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: suma**
      - ❑ Admite cero argumentos

        (+) ➔ 0
      - ❑ Un argumento

        (+ 3) ➔ 3
      - ❑ Dos argumentos

        (+ 3 4) ➔ *7*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: suma**
      - ❑ Tres o más argumentos

(+ 2.7 10 5.0 7) ➔ *24.7*

(+ 4 5 12 7 6) ➔ 34

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: suma**
      - ❑ Los argumentos pueden ser sub-expresiones

        (+ (+ 2.5 4) (+ 3.0 9)) ➔ *18.5*

      - ❑ Utilización de la forma “sangrada”

        (+

        (+ 2.5 4)

        (+ 3.0 9)

) ➔ *18.5*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: suma**
      - ❑ Equivalencia

(+ (+ 2.5 4) (+ 3.0 9)) ➔ *18.5*

(+ 2.5 4 3.0 9) ➔ *18.5*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: suma**
      - ❑ Observación:
        - **Siempre** hay que separar el operador de los argumentos usando espacios en blanco

(+3 7 5)

➔ Error

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: producto**
      - ❑Cero argumentos

        ( )* ➔ *1*

      - ❑ Un argumento

        (2)* ➔ *2*

      - ❑Dos argumentos

        (2 3)* ➔ *6*

      - ❑ Tres o más argumentos

        (3 4.0 1.5 2)* ➔ *36.0*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: producto**

❑ Expresiones anidadas

((* 3.5 3) (* 7 2.5))* ➔ *119.75*

❑ Expresiones “sangradas”

```
(*
        (* 3.5 3)
        (* 7 2.5)
)                          ➔ 119.75
```

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: producto**
      - ❑ Equivalencia

((* 3.5 3) (* 7 2.5))* ➔ *119.75*

(3.5 3 7 2.5)* ➔ *119.75*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: diferencia**
      - ❑ No admite cero argumentos

        (-) ➔ Error
      - ❑ Un argumento

        (- 3) ➔ -3
      - ❑ Dos argumentos

        (- 10 3) ➔ *7*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: diferencia**
      - ❑ Tres o más argumentos

        (- 10 2 3 4) ➔ *1*

      - ❑ Equivalente a

        (- 10 (+ 2 3 4)) ➔ *1*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: diferencia**
      - ❑ Anidamiento de expresiones

(- (- 100 25) (- 3 12)) ➔ *84*

(-
(- 100 25)
(- 3 12)
) ➔ *84*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: división**
      - ❑ No admite cero argumentos

        (/) ➔ Error

      - ❑ Un argumento

        (/ 9) ➔ $\frac{1}{9}$

        (/ 9.0) ➔ 0.1111111

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: división**
      - ❑ Dos argumentos

(/ 10 2) ➔ *5*

(/ 10 3) ➔ *3* $\frac{1}{3}$

Nota: representa una suma $3+\frac{1}{3}$

(/ 10 3.) ➔ *3.333333*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: división**
      - ❑ Tres o más argumentos

        (/ 60 3 4) ➔ *5*

      - ❑ Equivalente a

        (/ 60 (3 4))* ➔ *5*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos:**
      - ❑ Anidamiento de expresiones

```
(+
        (* 2
            (- 7 4)
        )
          (/  10 2)
)
➔ 11
```

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: expresiones con variables**

*(**define** pi 3.141592)*
pi ➔ *3.141592*

*(**define** radio 10)*
radio ➔ *10*

(pi (* radio radio))* ➔ *314.1592*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: expresiones con variables**

```
(define area
        (* pi (* radio radio))
)

area    ➔ 314.1592
```

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos: expresiones con variables**

```
(/
        (* 4 pi (* radio radio radio))
        3
)       ➔ 4188.7893
(define volumen
          (/
                (* 4 pi (* radio radio radio))
                3
           )
)
volumen       ➔ 4188.7893
```

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos:** *funciones predefinidas*

*(**define** otro_pi (/ 355.0 113.0))*

otro_pi ➔ *3.1415929*

*(**define** pi (acos -1.0))*

pi ➔ *3.141592653589793*

2. Expresiones

- Tipos de expresiones
  - **Operadores, funciones o procedimientos**
    - **Ejemplos:** *funciones predefinidas*

(sqrt 2) → 1.4142135623730951

(exp 1) → 2.718281828459045

(expt 2 3) → 8

2. Expresiones

- Tipos de expresiones
  - Literales
  - Variables
  - Operadores, funciones o procedimientos
  - **Formas especiales**

2. Expresiones

- Tipos de expresiones
  - **Formas especiales**
    - Expresiones que tienen al principio una palabra clave del lenguaje *scheme*.

2. Expresiones

- Tipos de expresiones
  - **Formas especiales**
    - Tipos
      - ❑Crean un nuevo contexto de ámbito léxico
        - ✓ ***do***, ***lambda***, ***let***, ***let**** y ***letrec***
      - ❑Controlan la ejecución
        - ✓ ***begin***, ***case***, ***cond***, ***do***, ***if***, ***and***, ***or***
      - ❑ Manejo de errores
        - ✓ assert, bkpt y error.
      - ❑ Extensiones sintácticas
        - ✓ macro y syntax

Índice

3. Funciones o procedimientos definidos por el programador

- Definición de funciones o procedimientos
- Evaluación mediante el modelo de sustitución
- Definiciones internas y estructuras de bloque

3. Funciones o procedimientos definidos por el programador

- Definición de funciones o procedimientos
- Evaluación mediante el modelo de sustitución
- Definiciones internas y estructuras de bloque

3. Funciones o procedimientos definidos por el programador

- Definición de funciones o procedimientos
  - Asocia una expresión compuesta a un identificador dentro de un **contexto léxico o estático**
  - **Sintaxis**

    *(**define** (<nombre> [<argumentos>])*
    <cuerpo>
)

    - donde
      - <*nombre*>: identificador de Scheme
      - <*argumentos*>:
        - ✓parámetros de la función (opcionales)
      - <*cuerpo*>
        - ✓*declaración de variables*
        - ✓*declaración de funciones internas*
        - ✓una o más expresiones

3. Funciones o procedimientos definidos por el programador

- Definición de funciones o procedimientos

  - **Ejemplos**

```
(define (cuadrado x)
        (* x x)
)
```

    - Se invoca el procedimiento "cuadrado"

(cuadrado 2) ➔ *4*

(cuadrado 3.0) ➔ *9.0*

(cuadrado (+ 3 6)) ➔ *81*

(cuadrado (cuadrado 4)) ➔ *256*

3. Funciones o procedimientos definidos por el programador

- Definición de funciones o procedimientos
  - **Ejemplos**
    - Llamando a la función “cuadrado” con variables

```
(define x 3)
(define y 4)

(+
        (cuadrado x)
        (cuadrado y)
)
➔ 25
```

3. Funciones o procedimientos definidos por el programador

- Definición de funciones o procedimientos
  - **Ejemplos**
    - Uso de "cuadrado" para definir otra función

```
(define (suma-de-cuadrados x y)
        (+
                (cuadrado x)
                (cuadrado y)
        )
)

(suma-de-cuadrados 3 4)   ➔ 25
```

3. Funciones o procedimientos definidos por el programador

- Definición de funciones o procedimientos
  - **Ejemplos**
    - Uso de "suma-de-cuadrados" para definir otra función:

```
(define (f z)
        (suma-de-cuadrados (* z 2) (/ z 2))
)

(f (* 2 5))          ➔ 425
```

3. Funciones o procedimientos definidos por el programador

- Definición de funciones o procedimientos
- Evaluación mediante el modelo de sustitución
- Definiciones internas y estructuras de bloque

3. Funciones o procedimientos definidos por el programador

- Evaluación mediante el método de sustitución
  - **Modelos**
    - Evaluación según el orden de aplicación *(applicative-order evaluation)*
    - Evaluación según el orden normal *(normal-order evaluation)*

3. Funciones o procedimientos definidos por el programador

- Evaluación mediante el método de sustitución
  - **Evaluación en el orden de aplicación: pasos**
    - Evalúa los argumentos y después aplica la función
      1. Se evalúan los valores de los argumentos (en un orden no determinado).
      2. Se aplica la función a los valores de los argumentos.

3. Funciones o procedimientos definidos por el programador

• Evaluación mediante el método de sustitución

○ Evaluación en el orden de aplicación: ejemplo (1/3)

- **Evaluación de *(f (* 2 5))***

1. Evaluación del valor de z: 10

2. Aplicación de f a 10
   (suma-de-cuadrados (10 2) (/ 10 2))*

3. Evaluación de los parámetros de suma-de-cuadrados (en un orden no determinado)
   (suma-de-cuadrados 20 5)

…

3. Funciones o procedimientos definidos por el programador

- Evaluación mediante el método de sustitución
  - **Evaluación en el orden de aplicación: ejemplo (2/3)**
    - **Evaluación de *(f (* 2 5))***

4. Aplicación del suma-de-cuadrados a los parámetros 20 y 5
   (+ (cuadrado 20) (cuadrado 5))

5. Aplicación de la función “cuadrado” a sus argumentos
   (+ (20 20) (* 5 5))*

…

3. Funciones o procedimientos definidos por el programador
   - Evaluación mediante el método de sustitución
     - **Evaluación en el orden de aplicación: ejemplo (3/3)**
       - **Evaluación de *(f 10)***

6. Evaluación de las expresiones

(+ 400 25)

425

3. Funciones o procedimientos definidos por el programador

- Evaluación mediante el método de sustitución
  - **Evaluación en el orden de aplicación**
    - **Observaciones**
      - ❑ Este modelo permite comprender **cómo** se **aplica** una **función**.
      - ❑ Se utiliza un contexto o **ámbito local** para la evaluación de los parámetros formales.

3. Funciones o procedimientos definidos por el programador

- Evaluación mediante el método de sustitución
  - **Evaluación en el orden normal: pasos**
    - Se expande completamente y después se reduce
      1. Se **sustituye** cada función o argumento por expresiones cada vez más simples.
         - ✓ Se obtiene una expresión compuesta exclusivamente por operadores primitivos
      2. Se **evalúa** la expresión con los operadores primitivos.

3. Funciones o procedimientos definidos por el programador

- Evaluación mediante el método de sustitución
  - **Evaluación en el orden normal: ejemplo**
    - **Evaluación de** *(f (* 2 5))*

1. Sustitución de ***f*** y ***z***

(suma-de-cuadrados ((* 2 5) 2) (/ (* 2 5) 2))*

2. Sustitución de “suma-de-cuadrados”

(+ (cuadrado ((* 2 5) 2)) (cuadrado (/ (* 2 5) 2))*

3. Sustitución de “cuadrado”

```
(+
   (*   (* (* 2 5) 2)   (* (* 2 5) 2))
   (*   (/ (* 2 5) 2)   (/ (* 2 5) 2))
)
```

3. Funciones o procedimientos definidos por el programador

- Evaluación mediante el método de sustitución
  - **Evaluación en el orden normal: ejemplo**
    - **Evaluación de** *(f (* 2 5))*

4. Evaluación de las expresiones

```
(+
    (*   (*  10 2)  (* 10 2))
    (*   (/  10 2)  (/  10 2))
 )
```

(+ (20 20) (* 5 5))*

(+ 400 25)

425

3. Funciones o procedimientos definidos por el programador

- Evaluación mediante el método de sustitución
  - **Comparación del orden normal y el orden aplicativo**
    - El proceso es diferente pero el resultado es el mismo
    - El "orden normal"
      - ❑ Es más ineficiente, porque realiza más operaciones
        - ✓ Por ejemplo: evalúa cuatro veces la expresión *(* 2 5)*
      - ❑ Necesita más recursos de memoria

3. Funciones o procedimientos definidos por el programador

- Evaluación mediante el método de sustitución
  - **Variantes de la evaluación en orden normal**
    - Evaluación retardada (*delayed evaluation*)
      - ❑ Es útil si se manejan “estructuras infinitas de datos”
    - Evaluación de llamada por necesidad (*call by need evaluation*)
      - ❑ Impide evaluaciones múltiples que surgen en la evaluación en orden normal estricta.

3. Funciones o procedimientos definidos por el programador

- Definición de funciones o procedimientos
- Evaluación mediante el modelo de sustitución
- Definiciones internas y estructuras de bloque

3. Funciones o procedimientos definidos por el programador

- Definiciones internas y estructuras de bloque
  - **Contexto o ámbito léxico de una función**
    - Regla de **anidamiento** más cercano
      - ❑ Si un identificador aparece en una función entonces su significado se determina
        1. Comprobando si es una **variable local** o **función interna**
        2. O si es un **parámetro formal**
        3. O buscando en el **contexto léxico** más próximo que **engloba** a la función.

3. Funciones o procedimientos definidos por el programador

- Definiciones internas y estructuras de bloque
  - **Contexto o ámbito léxico de una función**
    - **Ejemplo 1**

```
(define (prueba x)
    (* 2 x)
)
(prueba 3) ➔ 6
```

3. Funciones o procedimientos definidos por el programador
   - Definiciones internas y estructuras de bloque
     - **Contexto o ámbito léxico de una función**
       - **Ejemplo 2**

```
(define (prueba x)
    ;; Variable local
    (define x 9)
    ;; Uso de la variable local
    (* 2 x)
)
(prueba 3) ➔ 18
```

3. Funciones o procedimientos definidos por el programador

- Definiciones internas y estructuras de bloque
  - **Contexto o ámbito léxico de una función**
    - **Ejemplo 3**

```
(define (area radio)
        (define pi (acos -1.0))

        (* pi radio radio)
)
(area 1) ➔   3.1415929203539825
```

3. Funciones o procedimientos definidos por el programador
   - Definiciones internas y estructuras de bloque
     - **Contexto o ámbito léxico de una función**
       - **Ejemplo 4**

```
(define pi (acos -1.0))

(define (area radio)
        (* pi radio radio)
)

(area 1) ➔    3.1415929203539825
```

3. Funciones o procedimientos definidos por el programador

- Definiciones internas y estructuras de bloque
  - **Contexto o ámbito léxico de una función**
    - **Funciones anidadas**
      - ❑ Una función también puede **incluir** la definición de otras **funciones internas** o auxiliares.
      - ❑ Las funciones internas o auxiliares sólo son **accesibles dentro** del **contexto** en el que han sido definidas.

3. Funciones o procedimientos definidos por el programador

- Definiciones internas y estructuras de bloque
  - **Contexto o ámbito léxico de una función**
    - **Funciones anidadas**

❑ **Ejemplo 5**

```
(define (hipotenusa cateto1 cateto2)
        (define (cuadrado x) (* x x))
        (sqrt (+
                (cuadrado cateto1)
                (cuadrado cateto2)
                 )
        )
)
```

(hipotenusa 3 4) ➔ *5*

(cuadrado 2) ➔ Error

3. Funciones o procedimientos definidos por el programador

- Definiciones internas y estructuras de bloque
  - **Contexto o ámbito léxico de una función**
    - **Funciones no anidadas**

❑ **Ejemplo 6**

```
(define (cuadrado x) (* x x))
(define (hipotenusa cateto1 cateto2)
        (sqrt (+
                (cuadrado cateto1)
                (cuadrado cateto2)
                 )
        )
)
```

(hipotenusa 3 4) ➔ *5*

(cuadrado 2) ➔ 4

3. Funciones o procedimientos definidos por el programador
   - Definiciones internas y estructuras de bloque
     - **Contexto o ámbito léxico de una función**
       - **Funciones anidadas**

❑ **Ejemplo 7**

```
(define (volumen radio)
        ;; variable local
        (define pi (acos -1.0))
        ;; función auxiliar
        (define (cubo x) (* x x x))
        ;;
        (/ (* 4 pi (cubo radio)) 3)
)

(volumen 1)  ➔ 4.18879056047198
```

Índice

4. Procedimientos anónimos: la forma especial “lambda”

- Descripción
  - Permite crear una función sin nombre
  - **Utilidad**
    - Funciones auxiliares **temporales**
      - **Evita** definir funciones que nunca serán utilizadas fuera del contexto en el que han sido declaradas.
    - Funciones con **parámetros opcionales**
      - Nota: se explicará en el tema 5.

4. Procedimientos anónimos: la forma especial “lambda”

- Descripción
  - **Sintaxis**

    *(**lambda** ([<argumentos>]) <cuerpo>)*

    - donde
      - ❑ *<argumentos>*
        - ✓parámetros de la función (opcionales)
      - ❑ *<cuerpo>*
        - ✓*declaración de variables*
        - ✓*declaración de funciones internas*
        - ✓una o más expresiones

4. Procedimientos anónimos: la forma especial “lambda”

- Descripción
  - **Equivalencia**

```
(define (cubo x)
        (* x x x)
)

(define cubo
        (lambda (x) (* x x x))
)
```

4. Procedimientos anónimos: la forma especial “lambda”

- Descripción
  - **Ejemplo:**

```
(
        (lambda (x y z)
                (* x y (+ 3 z))
        )
        2   3   4
)
➔ 42
```

4. Procedimientos anónimos: la forma especial “lambda”

- Descripción
  - **Ejemplo**

```
(define (volumen radio)
        ;; variable local
        (define pi (acos -1.0))
        ;;
        (/
                (* 4.0
                pi
                ((lambda (x) (* x x x)) radio)
                )
        3.0
        )
)
(volumen 1)  ➔ 4.18879056047198
```

4. Procedimientos anónimos: la forma especial “lambda”

- Descripción
  - **Ejemplo**
    - **No es obligatorio el uso de lambda**

```
(define (volumen radio)
        ;; variable local
        (define pi (acos -1.0))
        ;;
        (/
                (* 4
                pi
                (* radio radio radio)
                )
        3)
)
```

4. Procedimientos anónimos: la forma especial “lambda”

- Descripción
  - Se recomienda el uso de ***lambda*** si un **parámetro** está asociado a una **expresión compleja** que se **repite** muchas veces.

4. Procedimientos anónimos: la forma especial “lambda”

- Descripción
  - **Ejemplo**

```
(define (volumen circunferencia)
 (define pi (acos -1.0))
        (/
           (* 4
              pi
              ((lambda (x) (* x x x))
                        (/ circunferencia 2 pi)
                )
           )
          3
        )
)
```

4. Procedimientos anónimos: la forma especial “lambda”

- Descripción
  - **Ejemplo: variable local que calcula el radio**

```
(define (volumen circunferencia)
        (define pi (acos -1.0))
        (define radio (/ circunferencia 2 pi))
                (/
                   (* 4
                      pi
                      ((lambda (x) (* x x x))
                                  radio
                       )
                    )
                  3
                )
)
```

Índice

5. Definición de variables locales: la forma especial “let”

- Introducción
- Descripción
- Comparación entre “***let***” y “***define***”
- Forma especial “***let****”

5. Definición de variables locales: la forma especial “let”

- Introducción
- Descripción
- Comparación entre “***let***” y “***define***”
- Forma especial “***let****”

5. Definición de variables locales: la forma especial “let”

- Introducción
  - La forma especial ***lambda*** también se utiliza para definir variables auxiliares usadas como parámetros.
  - Estas variables auxiliares también pueden ser declaradas con la forma especial “let”.

5. Definición de variables locales: la forma especial "let"

- Introducción
  - **Ejemplo** (1/5)
    - Uso de "la fórmula de Herón" para calcular el área de un triángulo.

$$área = \sqrt{s(s-a)(s-b(s-c)}$$

    - donde
      - *a*, *b* y *c*: lados del triángulo
      - *s*: semi-perímetro = *(a + b + c) / 2*

5. Definición de variables locales: la forma especial “let”

- Introducción
  - **Ejemplo** (2/5)
    - Primera versión: variable local creada con ***define***

```
(define (Heron a b c)
        ;; variable local: semi-perímetro
        (define s (/ (+ a b c) 2.0))

        ;;
        (sqrt (* s (- s a) (- s b) (- s c)))
)
```

5. Definición de variables locales: la forma especial “let”

- Introducción
  - **Ejemplo** (3/5)
    - Segunda versión: función auxiliar

```
(define (Heron a b c)
        ;; función auxiliar
        (define (auxiliar s)
                (sqrt (* s (- s a) (- s b) (- s c))
        )

        ;; llamada a la función auxiliar
        (auxiliar (/ (+ a b c) 2.0)
)
```

5. Definición de variables locales: la forma especial “let”

- Introducción
  - **Ejemplo** (4/5)
    - Tercera versión: función anónima (lambda)

```
(define (Heron a b c)
        ( (lambda (s)
                  (sqrt (* s (- s a) (- s b) (- s c))
          )
          (/ (+ a b c) 2.0)
        )
)
```

5. Definición de variables locales: la forma especial "let"

- Introducción
  - **Ejemplo** (5/5)
    - Cuarta versión: forma especial "let"

```
(define (Heron a b c)
        (let    ;; variable local de let
                (
                        (s (/ (+ a b c) 2.0))
                )
                ;; cuerpo de let
                (sqrt (* s (- s a) (- s b) (- s c)))
        )
)
```

5. Definición de variables locales: la forma especial “let”

- Introducción
- Descripción
- Comparación entre “***let***” y “***define***”
- Forma especial “***let****”

5. Definición de variables locales: la forma especial “let”

- Descripción
  - **Forma especial *let***
    - Constructor de enlace de variables que extiende el contexto léxico actual.
    - Permite la definición de variables locales
    - Permite evaluar expresiones en un ámbito léxico específico.

5. Definición de variables locales: la forma especial “let”

- Descripción
  - **Sintaxis**

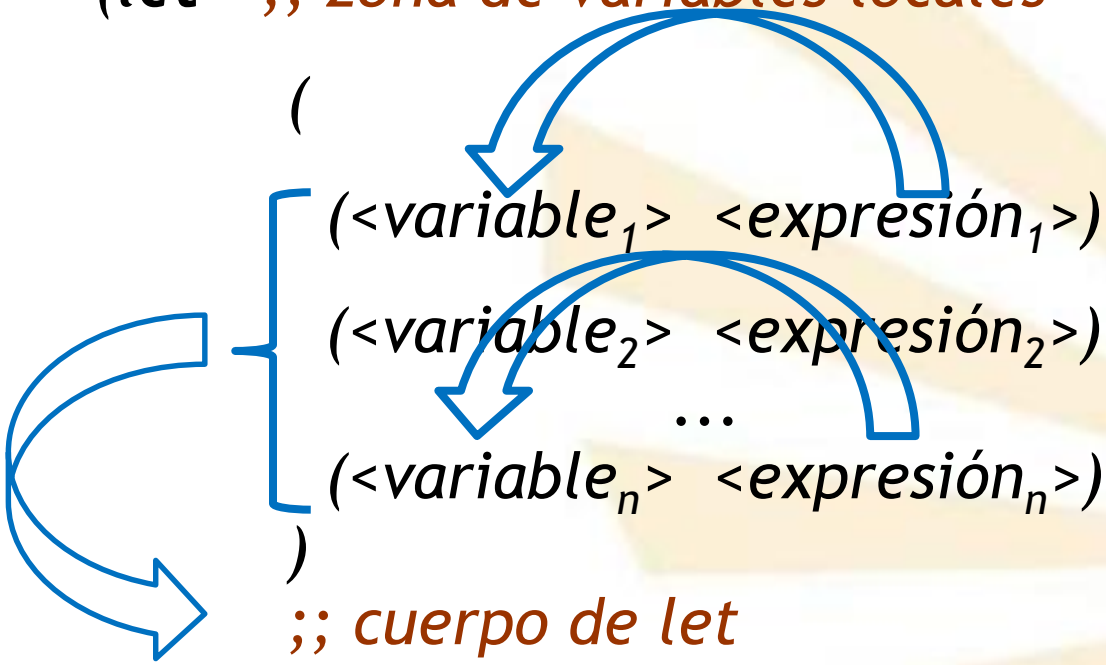
```
(let   ;; zona de variables locales
       (
         (<variable1>  <expresión1>)
         (<variable2>  <expresión2>)
                  ...
         (<variablen>  <expresiónn>)
       )
       ;; cuerpo de let
       <cuerpo>
)
```

5. Definición de variables locales: la forma especial “let”

- Descripción
  - **Semántica**
    - **Paso 1:** se evalúan las expresiones en un orden no determinado (“**evaluación en paralelo**”)
      - El **ámbito** de cada variable local se circunscribe al **cuerpo** de ***let*** y, por tanto, no incluye a las expresiones que asociadas a las variables.
    - **Paso 2:** el valor de cada expresión se **asigna** a la variable correspondiente.
    - **Paso 3:** evalúa el **cuerpo** en el contexto extendido que incluye las variables de ***let***.

5. Definición de variables locales: la forma especial "let"

- Descripción
  - **Sintaxis**



5. Definición de variables locales: la forma especial “let”

- Descripción
  - **Forma especial de “lambda” equivalente a “let”**

(
  (**lambda** (<*variable*$_1$> < *variable*$_2$> ... <*variable*$_n$>)
    <*cuerpo*>
)
    <*expresión*$_1$>
    <*expresión*$_2$>
    ...
    <*expresión*$_n$>
)

5. Definición de variables locales: la forma especial “let”

- Descripción
  - **Ejemplo** (1/5)
    - Se desea computar la función

$$f(x, y)= x\,(1 + x\,y)^2 + y\,(1 - y) + (1 + x\,y)\,(1 - y)$$

❑ Se utilizará dos variables locales: a y b.

✓ $a = 1 + x\,y$

✓ $b = 1 - y$

❑ quedando

✓ $f(x,y) = x\,a^2 + y\,b + a\,b$

5. Definición de variables locales: la forma especial “let”

- Descripción
  - **Ejemplo** (2/5)
    - Primera versión: uso de “***define***”

```
(define (f x y)
        ;; variables locales
        (define a (+ 1 (* x y)))
        (define b (- 1 y))
        ;;
        (+
                (* x (square a))
                (* y b)
                (* a b)
        )
)
```

5. Definición de variables locales: la forma especial “let”

- Descripción
  - **Ejemplo** (3/5)
    - Segunda versión: función auxiliar

```
(define (f x y)
        ;; función local
        (define (auxiliar a b)
        (+      (* x (square a))
                     (* y b)
                     (* a b)
                )
        )
        ;; llamada a la función local
        (auxiliar  (+ 1 (* x y)) (- 1 y))
)
```

5. Definición de variables locales: la forma especial "let"

- Descripción
  - **Ejemplo** (4/5)
    - Tercera versión: uso de lambda

```
(define (f x y)
        ( (lambda (a b)
                  (+
                        (* x (square a))
                        (* y b)
                        (* a b)
                  )
          )
          (+ 1 (* x y)) (- 1 y)
        )
)
```

5. Definición de variables locales: la forma especial "let"

- Descripción
  - **Ejemplo** (5/5)
    - Cuarta versión: uso de ***let***

```
(define (f x y)
        (let (
              (a   (+ 1 (* x y)) )
              (b   (- 1 y) )
             )
             (+ (* x (square a))
                (* y b)
                (* a b)
             )
        )
)
```

5. Definición de variables locales: la forma especial “let”

- Introducción
- Descripción
- Comparación entre “***let***” y “***define***”
- Forma especial “let*”

5. Definición de variables locales: la forma especial "let"

- Comparación entre "***let***" y "***define***"

1. Ámbito de las variables

- El ámbito creado por **define** abarca **todo el contexto** en el que puede ser ejecutado.

- ***let*** (***lambda***) crea un ámbito específico para las variables locales
  - El **ámbito** de una variable declarada con ***let*** es el **cuerpo** de ***let***.
  - ***let*** permite construir expresiones que enlazan variables tan localmente como sea posible.

5. Definición de variables locales: la forma especial "let"

- Comparación entre "***let***" y "***define***"
  - **Ejemplo**
    - Supóngase que x (externa) tiene el valor 5 y que se evalúa la siguiente expresión

```
(+  ;; primer sumando = 33
    (let (
          (x 3)
          )
          (+ x (* x 10))
    )
    ;; segundo sumando = 5
   x
)
```

➔ *38*

5. Definición de variables locales: la forma especial "let"

- Comparación entre "***let***" y "***define***"

2. Asignación de valores iniciales

- ***define***
  - ✓ Realiza una evaluación **secuencial** de las expresiones de las variables
  - ✓ Una expresión asociada a una variable **puede** usar los valores de las variables anteriores

- ***let***
  - ✓ Evalúa en **paralelo** (o no determinado) las expresiones de las variables
  - ✓ Una expresión de una variable no puede usar los valores de las variables anteriores.

5. Definición de variables locales: la forma especial "let"

- Comparación entre "***let***" y "***define***"

  - **Ejemplo** (1/2)
    - En un contexto externo, "x" vale 2
    - Se evalúa la siguiente expresión

```
(let
        ;; variables locales
        (
          (x 3)
          (y (+ x 2))  ;; "y" toma el valor 4
        )
        ;; cuerpo de let
        (* x y)
)
```

➔ 12

5. Definición de variables locales: la forma especial "let"

- Comparación entre "***let***" y "***define***"

  - **Ejemplo** (2/2)
    - Por el contrario, la siguiente secuencia

```
(define x 3)
(define y (+ x 2))

;;
(* x y)
```

➔ 15

5. Definición de variables locales: la forma especial “let”

- Introducción
- Descripción
- Comparación entre “***let***” y “***define***”
- Forma especial “***let****”

5. Definición de variables locales: la forma especial "let"

- Forma especial "***let****"
  - **Descripción**
    - Variante de la forma especial **let** que sí permite evaluar las variables locales de forma secuencial o consecutiva.

  - **Reglas de contexto**
    - El ámbito de cada variable local se extiende desde su inicialización hasta el cuerpo de **let***.
    - Por tanto,
      - la evaluación de la segunda variable se realiza en un contexto en el cual la primera variable enlace es visible,
      - y así sucesivamente.

5. Definición de variables locales: la forma especial “let”

- Forma especial “***let****”*
  - **Ejemplo**

```
(let* (
        (x 3)
        (y (+ x 2))
      )
        ;;
        (* x y)
 )
```

➔ 15

5. Definición de variables locales: la forma especial "let"

- Forma especial "***let****"*

  - **Sintaxis:** similar a la de ***let***

```
(let*   ;; zona de variables locales
                (
                  (<variable1>  <expresión1>)
                  (<variable2>  <expresión2>)
                                ...
                  (<variablen>  <expresiónn>)
                )
                ;; cuerpo de let*
                <cuerpo>
        )
```

5. Definición de variables locales: la forma especial "let"

- Forma especial "***let****"*
  - **Ejemplo: *let* y *let** (1/2)

```
(let ( ;; variables locales de let
         (x 2)
         (y 3)
       )
       ;; cuerpo de let
       (let* (;; variables locales de let*
                (x 7)
                (z (+ x y))
              )
              ;; cuerpo de let*
              (* z x)
         )
)
➔ 70
```

5. Definición de variables locales: la forma especial "let"

- Forma especial "***let****"*
  - **Ejemplo: *let* y *let** (2/2)**

```
(let ( ;; variables locales de let
         (x 2)
         (y 3)
       )
       ;; cuerpo de let
       (let* (;; variables locales de let*
                (z (+ x y))
                (x 7)
              )
              ;; cuerpo de let*
              (* z x)
         )
)
➔ 35
```

UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA

DEPARTAMENTO DE
INFORMÁTICA Y ANÁLISIS NUMÉRICO

PROGRAMACIÓN DECLARATIVA

INGENIERÍA INFORMÁTICA

CUARTO CURSO

PRIMER CUATRIMESTRE

Tema 2.- Expresiones y funciones