# Lenguajes de programación

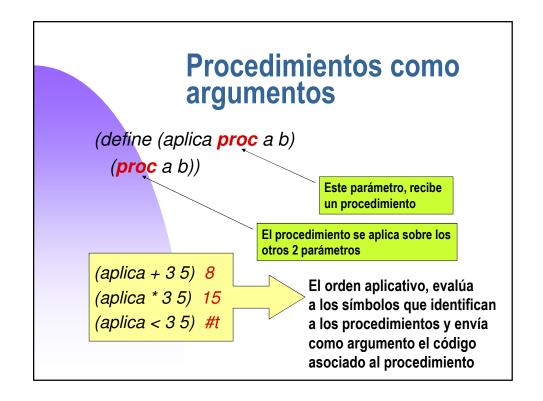
Funciones de Orden Superior y la Forma Especial Lambda

# **Objetos de primera clase (first-class)**

- Son aquellos elementos de un lenguaje de programación que:
  - se pueden pasar como argumentos a los parámetros.
  - se pueden generar como resultado de una función.
  - pueden formar parte de una estructura de datos.
- Ejemplos conocidos: datos atómicos y listas de datos.

#### Funciones en Lenguajes Funcionales

- Las FUNCIONES son objetos de primer clase, lo que significa que:
  - se pueden pasar como argumentos a los parámetros !!
  - se pueden generar como resultado de una función !!
  - pueden formar parte de una estructura de datos !!



### Procedimientos como resultado de un procedimiento

(if (< 1 2) + \*) 
$$\rightarrow$$
 # ((if (< 1 2) + \*) 3 5)  $\rightarrow$  8

(define (tipo n)  
(cond ((= n 1) +)  
((= n 2) -)  
((= n 3) \*)  
((= n 4) /)))  
((tipo 2) 35) 
$$\rightarrow$$
 -2

El resultado de esta llamada es un procedimiento que se puede aplicar

## Listas de procedimientos

 $((cadddr LP) 3 5) \rightarrow 0.6$ 

La evaluación de los argumentos genera el código asociado a cada procedimiento, formando una lista de códigos

#### Forma especial LAMBDA

- Sirve para definir un procedimiento sin nombre.
- El nombre "lambda" es heredado del cálculo lambda de Alonso Church.
- Formato:

(lambda (parámetros) cuerpo)

Lista de símbolos

Expresiones (código del procedimiento)

### Definición de procedimientos con nombre

 La definición de procedimientos que hemos utilizado hasta ahora, es en realidad una definición con la forma especial lambda.

Esta será la forma en que a partir de ahora definiremos procedimientos

(define (cuadrado x) (\* x x))

equivale a:

(define cuadrado (lambda (x) (\* x x)))

## Aplicación de la forma especial LAMBDA

- Para construir procedimientos "en línea", sin necesidad de que tengan asociado un nombre.
- Ejemplos:
  - > (lambda (x) (\* x x))
    - → #procedure>
  - $> ((lambda (x) (*x x)) 3) \rightarrow 9$

# Calculo lambda y la forma especial lambda

Relación directa:

$$(\lambda x . + x 3) 5$$



((lambda (x) (+ x 3)) 5)

### Predicados relacionados con procedimientos

- (primitive? arg)
  - Verifica si su argumento es un procedimiento primitivo (un procedimiento pre-construido en un lenguaje de bajo nivel)
- (procedure? arg)
  - Verifica si su argumento es una procedimiento

#### Funciones de Orden Superior (FOS)

- Funciones que hacen al menos una de las siguientes cosas:
  - Reciben una o más funciones como entrada
  - 2. Devuelven una función como salida
- Aprovechan que las funciones son objetos de primera clase para resolver problemas en forma eficiente.

### **Utilidad de las FOS:** Patrones de recursión

¿Cuál es la diferencia entre las dos funciones?

### **Utilidad de las FOS:** Patrones de recursión

Abstracción del operador y del resultado en caso base:

#### **Utilidad de las FOS:** Patrones de recursión Muchas funciones en una: (define fold (lambda (op init l) (if (null? 1) init ( op (car l) (fold op init (cdr l))))) (**define** append (define suma (lambda (l) (lambda (l1 12) (fold + 0 1)))(fold cons 12 11))) (define mult (lambda (l) (fold \* 1 1)))

# Utilidad de las FOS: Procedimientos Generadores

- Dado que la forma especial lambda permite definir procedimientos "en línea"...
- ¿Qué pasa si el cuerpo de un procedimiento se define con la forma especial lambda?...
- Este procedimiento, al evaluarse, generará como resultado un procedimiento

#### Currificación

- Diferencia:
  - En el Cálculo Lambda las funciones solo tienen un argumento
  - En Scheme pueden tener varios
- Igualación:
  - Convertir una función de varios argumentos de forma que pueda ser llamada como una cadena de funciones de un argumento (Currificarla)

#### **Ejemplo**

Scheme (sin currificar):

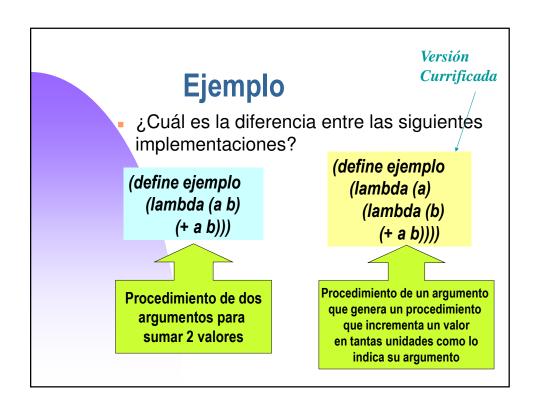
((lambda (x y) (+ x y)) 5 6)

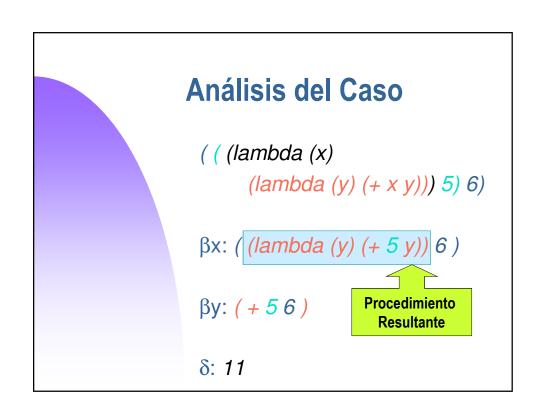
Scheme (currificada):

(((lambda (x) (lambda (y) (+ x y))) 5) 6)

λ-Calculus:

 $(\lambda x. \lambda y. + x y) 5 6$ 





#### Ejemplo de Aplicación

- Implementar un procedimiento que sirva para generar procedimientos que apliquen dos veces sobre un valor la función recibida como parámetro.
- ANALISIS:
  - ¿Cuál es el parámetro del procedimiento generador?
  - Cuál es el parámetro del procedimiento a generar?

#### **Implementación**

```
(define aplica-doble
(lambda (función)
(lambda (valor)
(función (función valor)))))
```

- > (define inc2 (aplica-doble add1))
- > (define quita2 (aplica-doble cdr))
- > (define eleva4a (aplica-doble (lambda (x) (\* x x)))