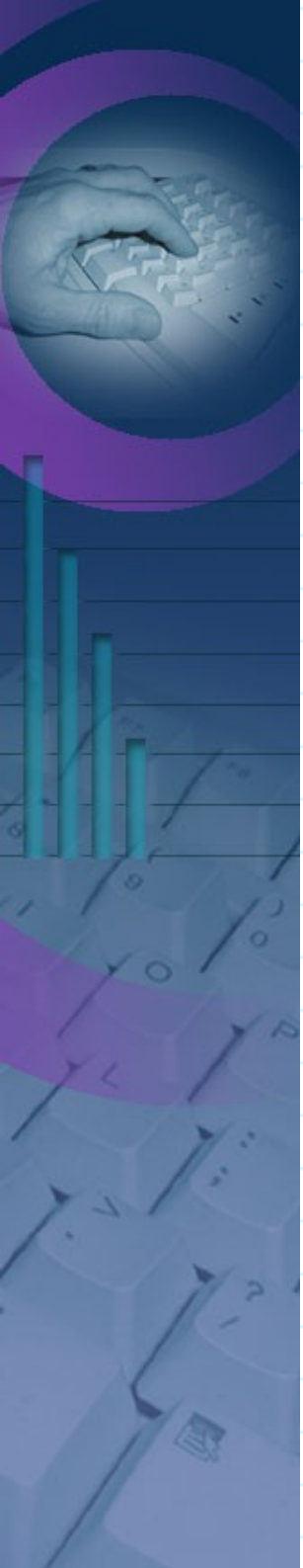


Variables y Expresiones Let

- La forma sintáctica Let incluye pares de *variables-expresiones* junto con una secuencia de expresiones que representan el cuerpo del Let

(let ((var1 val) [(var2 val)...]) exp1 exp2 ...)



Variables y Expresiones Let

- También se usa para simplificar expresiones
 - $(+ (* 4 4) (* 4 4)) \Rightarrow 32$
 - $(\text{let } ((a (* 4 4)))$
 $(+ a a)) \Rightarrow 32$
 - $(\text{let } ((\text{list1 } '(a b c)) (\text{list2 } '(d e f))))$
 $(\text{cons } (\text{cons } (\text{car list1}) (\text{car list2}))$
 $(\text{cons } (\text{car } (\text{cdr list1})) (\text{car } (\text{cdr list2}))))))$
 $\Rightarrow ((a . d) b . e)$



Variables y Expresiones Let

- Es posible anidar Lets
 - ```
(let ((a 4) (b -3))
 (let ((a-squared (* a a))
 (b-squared (* b b)))
 (+ a-squared b-squared)))
```

 → 25
  - ```
(let ((x 1))  
  (let ((x (+ x 1)))  
    (display x)))
```

 → 2



Expresiones Lambda

- Lambda permite crear un nuevo procedimiento
- Expresión general:
 - $(\text{lambda } (\text{var } \dots) \text{exp1 exp2 } \dots)$
 - Ejemplo: $(\text{lambda } (x) (+ x x)) \Rightarrow \#<\text{procedure}>$
- Uso:
 - $((\text{lambda } (x) (+ x x)) (* 3 4)) \Rightarrow 24$



Expresiones Lambda

- Como los procedimientos son objetos los podemos asignar a variables
 - ```
(let ((double (lambda (x) (+ x x))))
 (list (double (* 3 4))
 (double (/ 99 11))
 (double (- 2 7)))) ⇒ (24 18 -10)
```
  - ```
(let ((double-cons (lambda (x) (cons x x))))  
  (double-cons 'a)) ⇒ (a . a)
```



Definiciones de alto nivel

- Las definiciones de **Alto Nivel** son vistas desde todos los procedimientos
- se declaran a partir de la cláusula *define*
 - $(\text{define double-any}$
 $(\text{lambda (f x)}$
 $(f\ x\ x)))$
 - $(\text{double-any } +\ 10) \Rightarrow 20$
 - $(\text{double-any cons 'a}) \Rightarrow (a . a)$



Definiciones de Alto Nivel

- se pueden utilizar para cualquier tipo de objetos, no solo procedimientos
(define sandwich "milanesa-tomate-y-lechuga")
sandwich \Rightarrow "milanesa-tomate-y-lechuga"



Definiciones de Alto Nivel

- Scheme provee abreviaturas llamadas `cadr` y `cddr` que son composiciones de
 - `(car (cdr list))`
 - `(cdr (cdr list))`
 - `(define cadr
 (lambda (x)
 (car (cdr x))))`
 - `(define cddr
 (lambda (x)
 (cdr (cdr x))))`



Expresiones Condicionales

- (if (<test>
 (<verdad>
 (<falso>
)
)
- Ej: (define abs
 (lambda (n)
 (if (< n 0)
 (- 0 n)
 n)
))



Expresiones Condicionales

- not: devuelve el inverso del parámetro dado
 - $(\text{not } \#t) \rightarrow \#f$
 - $(\text{not "false"}) \rightarrow \#f$
 - $(\text{not } \#f) \rightarrow \#t$
- or/and: realiza la comparación lógica y devuelve el resultado
 - $(\text{or}) \rightarrow \#f$
 - $(\text{or } \#t \#f) \rightarrow \#t$
 - $(\text{and } \#t \#f) \rightarrow \#f$



Expresiones Condicionales

- $=$, $<$, $>$, \leq , y \geq son todos predicados y responden a preguntas específicas sobre sus argumentos devolviendo un valor de verdad
- los nombres de los predicados normalmente finalizan en $?$ excepto los anteriores



Expresiones Condicionales

- `null?` : devuelve `#t` si el argumento es una lista vacía
 - `(null? ())` \rightarrow `#t`
 - `(null? 'abc)` \Rightarrow `#f`
 - `(null? '(x y z))` \Rightarrow `#f`



Expresiones Condicionales

- `eqv?`: requiere dos argumentos y devuelve `#t` si son equivalentes
 - `(eqv? 'a 'a) ⇒ #t`
 - `(eqv? 'a 'b) ⇒ #f`
 - `(eqv? #f #f) ⇒ #t`
 - `(eqv? #t #t) ⇒ #t`
 - `(eqv? #f #t) ⇒ #f`
 - `(eqv? 3 3) ⇒ #t`
 - `(eqv? 3 2) ⇒ #f`



Expresiones Condicionales

- Otros predicados son:
 - pair?
 - symbol?
 - number?
 - string?



Expresiones Condicionales

- cond:permite realizar múltiples test/acciones. Su forma general es la siguiente:
 - `(cond (test exp) ... (else exp))`
 - Ej: `(define sign
 (lambda (n)
 (cond
 ((< n 0) -1)
 ((> n 0) +1)
 (else 0))))`



Expresiones Condicionales

- ```
(define income-tax
 (lambda (income)
 (cond
 ((<= income 10000) (* income .05))
 ((<= income 20000) (+ (* (- income 10000) .08) 500.00))
 ((<= income 30000) (+ (* (- income 20000) .13) 1300.00))
 (else (+ (* (- income 30000) .21) 2600.00)))))
```
- $(\text{income-tax } 5000) \Rightarrow 250.0$
- $(\text{income-tax } 15000) \Rightarrow 900.0$
- $(\text{income-tax } 25000) \Rightarrow 1950.0$
- $(\text{income-tax } 50000) \Rightarrow 6800.0$





# Paradigma Funcional

continuará...