



Tecnologías de Programación

Programación Funcional



Definición

La Programación funcional es un paradigma de programación declarativa basado en la utilización de funciones matemáticas.

Sus orígenes se remontan al cálculo lambda (o λ -cálculo), una teoría matemática elaborada por Alonzo Church (1930) como apoyo a sus estudios sobre computabilidad.



Lenguajes Funcionales

- Categorías de Lenguajes:
 - **Puros**: tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial.
 - Ejemplo: Haskel, Miranda.
 - **Híbridos**: admiten secuencias de instrucciones o la asignación de variables.
 - Ejemplo: Scala, Lisp, Scheme, Ocaml y Standard ML.



Descripción informal

- Todas las expresiones representan funciones de un sólo argumento.
- Este argumento, a su vez, sólo puede ser una función de un sólo argumento.
- Todas las funciones son anónimas, y están definidas por expresiones lambda, que dicen que se hace con su argumento.
- La aplicación de funciones es asociativa a izquierda: $f\ x\ y = (f\ x)\ y$.



Tecnologías de Programación

Racket



Racket

- Introducción:
 - es un lenguaje funcional, derivado de LISP.
 - compacto con un alto grado de abstracción.
 - permite resolver problemas complejos con programas cortos y elegantes.



Sintaxis

- Palabras clave, variables y los símbolos son llamados Identificadores. Los identificadores pueden estar formados por:
 - [a-z]
 - [A-Z]
 - [0-9]
 - ? ! . + - * / < = > : \$ % ° & _ ~ @
 - Ejemplo: Hola, n, x, x3, ?\$&*!!!



Sintaxis

- Todos los identificadores deben estar delimitados por:
 - Un espacio en blanco
 - Comillas dobles (“)
 - Paréntesis
 - carácter de comentario (;)
- No hay límite de longitud
- No es case-sensitive:
 - Abc, abc, aBc son todos el mismo identificador



Sintaxis

- Las estructuras y las Listas se encierran entre paréntesis:
 - Ejemplo: (a b c) o (* (- x 2) y)
- () => Lista vacía
- Valores Booleanos:
 - #t → verdadero
 - #f → falso



Sintaxis

- Vectores:
 - comienzan con #(
 - Finalizan con)
 - Ejemplo: #(esto es un vector de símbolos)
- String: encerrados en “”
- Caracteres: precedidos por #\
 - Ej: #\a



Sintaxis

- Números:
 - Enteros: -123
 - Racionales: $1/2$
 - En punto flotante: 1.3
 - Notación Científica: $1e12$
 - Complejos en Notación Rectangular: $1.3-2.7i$
 - Complejos en Notación Polar: $-1.2@73$



Convención de Nombres

- Predicados finalizan en ?: retornan #t o #f
 - Ejemplo: eq?, zero?, string=?
- El nombre de la mayoría de los procedimientos de string, caracteres y vectores comienzan con el prefijo string-, char- y vector-
 - Ejemplo: string-append
- Conversión entre tipos de objetos se escriben como *tipo1->tipo2*.
 - Ej: *vector->list*



Interactuando con Racket

- Probar:
 - "hola"
 - 42
 - $22/7$
 - 3.141592653
 - +
 - (+ 76 31)
 - '(a b c d)



Interactuando con Racket

- Resultados:
 - `"hola"` \Rightarrow `"hola"`
 - `42` \Rightarrow `42`
 - `22/7` \Rightarrow `3 1/7`
 - `3.141592653` \Rightarrow `3.141592653`
 - `+` \Rightarrow `#<primitive: +>`
 - `(+ 76 31)` \Rightarrow `107`
 - `'(a b c d)` \Rightarrow `(a b c d)`



Interactuando con Racket

- Identifique que hacen las funciones:
 - `(car '(a b c))`
 - `(cdr '(a b c))`
 - `(cons 'a '(b c))`
 - `(cons (car '(a b c))
 (cdr '(d e f)))`



Interactuando con Racket

- Definir un procedimiento:
(define cuadrado
 (lambda (n)
 (* n n)))
- Y usarlo:
 - (cuadrado 5) \Rightarrow 25
 - (cuadrado -200) \Rightarrow 40000
 - (cuadrado 0.5) \Rightarrow 0.25
 - (cuadrado -1/2) \Rightarrow 1/4



Expresiones simples

- Notación prefija
 - $(+ 2 2) \Rightarrow 4$
 - $(+ (+ 2 2) (+ 2 2)) \Rightarrow 8$
 - $(- 2 (* 4 1/3)) \Rightarrow 2/3$
 - $(* 2 (* 2 (* 2 (* 2 2)))) \Rightarrow 32$
 - $(/ (* 6/7 7/2) (- 4.5 1.5)) \Rightarrow 1.0$



Expresiones simples

- Estructura de agregación: Listas (list)
 - `(quote (1 2 3 4 5))` → lista de números
 - `'("esto" "es" "una" "lista")` → lista de strings
 - `'(4.2 "hola")` → lista de múltiples tipos
 - `'((1 2) ("hola" "Racket"))` → lista de listas



Expresiones Simples

- Quote (') le dice a Racket que trate un identificador como símbolo y no como variable
- Los símbolos y variables en Racket son similares a los símbolos y variables en expresiones matemáticas y ecuaciones
 - En expresiones Matemáticas:
 - $1 - x \rightarrow$ pensamos en x como variable
 - En expresiones Algebraicas:
 - $X^2 - 2 \rightarrow$ pensamos en x como símbolo



Expresiones Simples

- Operadores de Listas
 - car: retorna el primer elemento de la lista
 - cdr (could-er): retorna el resto de la lista
 - $(\text{car } '(a\ b\ c)) \Rightarrow a$
 - $(\text{cdr } '(a\ b\ c)) \Rightarrow (b\ c)$
 - $(\text{cdr } '(a)) \Rightarrow ()$



Expresiones Simples

- Operadores de Listas: car – cdr
 - Resuelva:
 - (car (cdr '(a b c)))
 - (cdr (cdr '(a b c)))
 - (car '((a b) (c d)))
 - (cdr '((a b) (c d)))



Expresiones Simples

- Operadores de Listas: car – cdr
 - Resultados:
 - $(\text{car } (\text{cdr } '(a\ b\ c))) \Rightarrow b$
 - $(\text{cdr } (\text{cdr } '(a\ b\ c))) \Rightarrow (c)$
 - $(\text{car } '((a\ b)\ (c\ d))) \Rightarrow (a\ b)$
 - $(\text{cdr } '((a\ b)\ (c\ d))) \Rightarrow ((c\ d))$



Expresiones Simples

- Operadores de Listas:
 - cons: construye listas. Recibe dos argumentos. Usualmente el segundo es una lista y en ese caso retorna una lista
 - $(\text{cons } 'a \ '()) \Rightarrow (a)$
 - $(\text{cons } 'a \ '(b \ c)) \Rightarrow (a \ b \ c)$
 - $(\text{cons } 'a \ (\text{cons } 'b \ (\text{cons } 'c \ '())) \Rightarrow (a \ b \ c)$
 - $(\text{cons } '(a \ b) \ '(c \ d)) \Rightarrow ((a \ b) \ c \ d)$



Expresiones Simples

- Operadores de Listas:
 - cons: construye listas. Recibe dos argumentos. Usualmente el segundo es una lista y en ese caso retorna una lista
 - $(\text{cons } 'a \ '()) \Rightarrow (a)$
 - $(\text{cons } 'a \ '(b \ c)) \Rightarrow (a \ b \ c)$
 - $(\text{cons } 'a \ (\text{cons } 'b \ (\text{cons } 'c \ '())))) \Rightarrow (a \ b \ c)$
 - $(\text{cons } '(a \ b) \ '(c \ d)) \Rightarrow ((a \ b) \ c \ d)$



Expresiones Simples

- Operadores de Listas:
 - Resuelva:
 - (car (cons 'a '(b c)))
 - (cdr (cons 'a '(b c)))
 - (cons (car '(a b c))
 (cdr '(d e f)))
 - (cons (car '(a b c))
 (cdr '(a b c)))



Expresiones Simples

- Operadores de Listas:
 - Resultados:
 - $(\text{car } (\text{cons } 'a \ '(b \ c))) \Rightarrow a$
 - $(\text{cdr } (\text{cons } 'a \ '(b \ c))) \Rightarrow (b \ c)$
 - $(\text{cons } (\text{car } '(a \ b \ c))$
 $(\text{cdr } '(d \ e \ f))) \Rightarrow (a \ e \ f)$
 - $(\text{cons } (\text{car } '(a \ b \ c))$
 $(\text{cdr } '(a \ b \ c))) \Rightarrow (a \ b \ c)$



Expresiones Simples

- Cons: construye pares. Cuando el segundo parámetro es una lista devuelve una lista *Propia*.
 - Lista Propia: Una lista vacía es una lista propia, y toda lista cuyo **cdr** sea una lista propia es una lista propia.
 - Listas impropias: compuestas por pares donde se marca la separación de elementos por puntos.



Expresiones simples

- $(\text{cons } 'a \ 'b) \rightarrow (a . b)$
- $(\text{cdr } '(a . b)) \rightarrow b$
 - A diferencia de $(\text{cdr } '(a b)) \rightarrow (b)$
- $(\text{cons } 'a \ '(b . c)) \rightarrow (a b . c)$



Evaluado de Expresiones

- (procedimiento arg1 arg2 ...)
 - Buscar el valor de *procedimiento*
 - Buscar el valor de *arg1*
 - Buscar el valor de *arg2*
 -
 - Aplicar el valor de *procedimiento* a los valores de *arg1*, *arg2*, ...



Evaluado de Expresiones

- Ejemplo:
 - $(+ \ 3 \ 4)$
 - el valor de $+$ es el procedimiento *adición*
 - el valor de 3 es el número 3
 - el valor de 4 es el número 4
 - aplicando el procedimiento *adición* a los números 3 y 4 devuelve 7
 - pruebe: `((car (cdr (list + - * /))) 17 5)`