

Programación Funcional

Pedro O. Pérez M., PhD.

Implementación de métodos computacionales
Tecnológico de Monterrey

pperezm@tec.mx

07-2023

1 Introducción

- Lenguajes de programación funcional
- Estilo de Programación Funcional
- Scheme

Lenguajes de programación funcional

En la práctica, los lenguajes funcionales...

- Todo lo modelan con funciones: definiciones y llamadas, la secuencia es una composición de funciones.
- No manejan variables, sólo parámetros.
- No manejan asignación de valores.
- No utilizan iteraciones, sólo recursividad.
- Almacenan todo en listas encadenadas y con asociaciones dinámicas.
- Tratan a las funciones como a los datos: son argumentos, resultados, estructuras.

Conceptualmente, los lenguajes funcionales...

- Son **declarativos**:
 - ▶ Expresan qué resolver y no cómo resolverlo.
- Son de **muy alto nivel de abstracción**:
 - ▶ Alejados del modelo de Von Nuemann y apegados al pensamiento “natural” humano.
 - ▶ Requieren mayor esfuerzo de traducción y pueden consumir muchos recursos en la ejecución.
- Tienen **transparencia referencial**:
 - ▶ No hay efectos laterales en memoria que alteren el significado de un programa.
- Son **minimalistas**:
 - ▶ Fácil lectura, mantenimiento, **paralelización** y comprobación.

Los lenguajes funcionales...

- Existen desde **1958** con la creación de **LisP** (**List Processing**).
- Tienen diferentes grados de hibridez al combinarse con diversos paradigmas. **Haskell** es uno de los consideramos más puro de todos los lenguajes funcionales.
- Han adquirido mayor importancia y popularidad por sus ventajas en el desarrollo de aplicaciones de **Inteligencia Artificial** y **Ciencia de datos**.

Estilo de Programación Funcional

El estilo de la programación funcional se puede usar en los lenguajes imperativos.

```
int factorial (int n)
{ int fact, j;
  fact = 1;
  for (j = 2; j<=n; j++)
    fact = fact * j;
  return fact;
};
```

```
int factorial (int n)
{
  if (n == 0)
    return 1;
  else
    return (n*factorial(n-1));
};
```

Estilo imperativo:

- Uso de variables y asignaciones.
- Uso de ciclos (iteraciones).
- Posible efecto lateral si usáramos variables globales.

Estilo funcional:

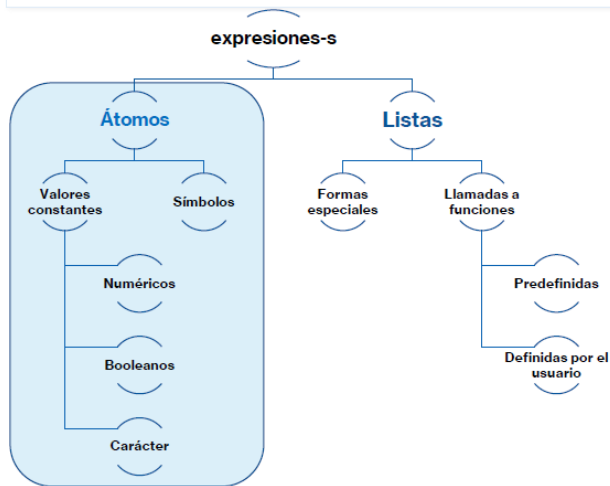
- No hay variables ni asignaciones.
- Uso de la recursividad.
- No hay efectos laterales al no usar variables globales.

- **Scheme** es un lenguaje funcional sencillo, minimalista, para aprender ágilmente el nuevo paradigma.
- Es un dialecto de **Lisp** creado en 1975 por Steele y Sussman (MIT).

Iniciando con Scheme...

- Toda su sintaxis se reduce al formato de una **lista** que es una **expresión-s**.
 - ▶ Las listas se delimitan con **paréntesis** y puede tener cero o más elementos.
 - ▶ Una lista es una expresión-s cuyos elementos son a su vez una expresión-s.
- Tiene pocas reglas **semánticas**, pues es un lenguaje tipado débil (no hay declaraciones de tipos de datos).

expresiones-s en Scheme



Átomos:

- La evaluación de una constante genera como resultado el valor de la propia constante.
- La evaluación de un símbolo genera como resultado el valor asociado a ese símbolo.
- Un símbolo es un identificador que se construye con cualquier caracter, excepto:

() [] { } ; , ' " # \

Funciones predefinidas

- Aritméticas: $+$, $-$, $*$, $/$, remainder, quotient, sqrt, etc.
- Relacionales: $<$, \leq , $>$, \geq , $=$.
- Lógicas: and, or, not.
- Predicados: positive?, zero?, even?, null?, char?, etc.
- Manejo de listas: car, cdr, cons, list.
- Manejo de funciones: map, apply.

Expresiones aritméticas en Scheme

- No hay operadores, sólo **funciones multiparámetro** que se aplican sobre los argumentos. **Importante: el formato es prefijo.**
- Ejemplos:
 - ▶ `(+)`
 - ▶ `(- 8)`
 - ▶ `(+ 3 4)`
 - ▶ `(+ 2 3 4 5 6 7 8)`
 - ▶ `(/ (+ 2 3) 5)`
 - ▶ `(+ (/ 2 3) 5)`
 - ▶ `(/ (- 7 3) (* 2 5))`

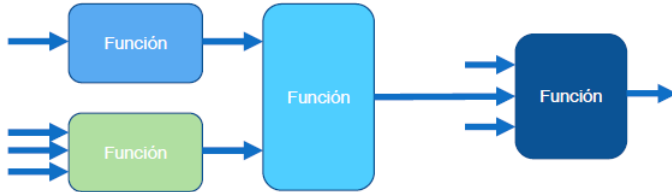
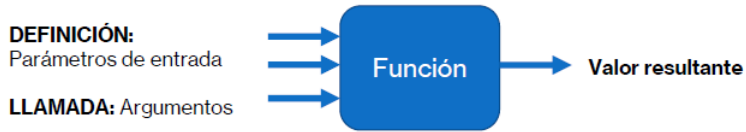
La forma especial define

- Sintaxis: (define símbolo expresión).
- Evalúa la expresión y el valor resultante lo asocia con el símbolo, de tal forma que el símbolo queda definido en el ambiente de trabajo.

Vamos a desarrollar las siguientes funciones:

- 1 La función `sum` recibe como entrada dos números, `a` y `b`. La función regresa la suma de ambos.
- 2 La función `area-of-triangle` recibe como entrada la base y altura de un triángulo. La función regresa el área del triángulo.

En el paradigma funciona, todo es una función...



Vamos a desarrollar las siguientes funciones:

- 1 La función `area-of-ring` recibe como entrada el radio interno y externo del anillo. La función regresa el área del anillo.
- 2 La función `volume-of-cylinder` recibe como entra el radio u la altura de un cilindro. La función regresa el volumen del mismo.

Revisa la actividad en Canvas.

La forma especial quote

- Sintaxis: (quote símbolo).
- Sirve para NO evaluar el símbolo, generando como resultado el propio símbolo. Útil para manejar a los símbolos como datos.
- Puede abreviarse con una comilla y sin necesidad de utilizar a la lista.
- (quote abc) es equivalente a 'abc.

La forma especial if

- Sintaxis: (if predicado if-true if-false).
- Se evalúa el predicado, si su valor es verdadero se evalúa la expresión consecuente; si es falso, se evalúa la expresión alternativa.
- Un **predicado** es una expresión que genera un valor booleano al evaluarse.
- A este tipo de evaluación se le conoce como **evaluación floja (lazy evaluation)**, pues no se evalúa lo que no es necesario.

Vamos a desarrollar las siguientes funciones:

- 1 La función `max2` recibe como entrada dos números, `a` y `b`. La función regresa el mayor de ambos números.
- 2 La función `max3` recibe como entrada tres números, `a`, `b` y `c`. La función regresa el mayor de los tres números.

La forma especial cond

- Sintaxis: $(\text{cond } (\text{predicado}_1, \text{expresion}_1), (\text{predicado}_2, \text{expresion}_2), \dots, (\text{predicado}_n, \text{expresion}_n), (\text{else } \text{expresion}_n))$.
- Forma de evaluación: evalúa la primera secuencia de expresiones cuyo predicado sea verdadero; si ningún predicado se cumple, evalúa la expresión del else

Ejemplos

Vamos a desarrollar las siguientes funciones:

- ➊ La función **max3** recibe como entrada tres números, a , b y c . La función regresa el mayor de los tres números.
- ➋ la función **interest** recibe como entrada el saldo de una cuenta bancaria de un banco. El banco paga un 4 % fijo para saldos de hasta \$1000, un 4.5 % fijo anual para saldos de hasta \$5000 y un 5 % fijo para saldos de más de \$5000.
- ➌ La función **how-many** recibe como entrada los coeficientes, a , b y c , de una ecuación cuadrática. La función determina cuántas soluciones tiene la ecuación. Asumiendo que a no es 0, la ecuación tiene:
 - ▶ 2 soluciones, si $b^2 > 4ac$.
 - ▶ 1 solución, si $b^2 = 4ac$.
 - ▶ 0 soluciones, si $b^2 < 4ac$.

Revisa la actividad en Canvas.

Recursividad en Scheme

- Ya conocemos las herramientas necesarias del lenguaje:
 - ▶ Definición y llamadas de funciones.
 - ▶ Decisiones con las formas especiales **if** y **cond**.
- Lo importante es **desarrollar el pensamiento recursivo** para la solución de problemas.
- Esto va más allá del uso de lenguaje, es un cambio de paradigma mental.

Para pensar recursivamente:

- Paso 1.
 - ▶ Analizar cuál es el **caso más simple o pequeño** del problema que se quiere resolver.
 - ▶ Este caso debe de tener una **solución clara y directa, no recursiva**.
 - ▶ Este caso se considera el **caso base** de la recursividad, y determina una **condición de salida** de la repetición implícita que se da en la recursividad.
- Paso 2.
 - ▶ Analizar cómo se resuelve el **problema general, suponiendo** que ya se tiene “algo” que resuelve el **siguiente caso más pequeño del problema**.
 - ▶ Este caso plantea la **solución recursiva** del problema.
 - ▶ La solución al siguiente caso más pequeño, se programa con la **llamada recursiva**, que se integra a la solución general del caso.

Vamos a desarrollar las siguientes funciones:

- 1 La función que obtiene la sumatoria desde 0 hasta n .
- 2 La función que despliega n veces el letrero “hola”.
- 3 La función que despliega la secuencia desde n hasta 1.
- 4 La función que cuenta la cantidad de dígitos de un número entero.

Revisa la actividad en Canvas.