Introducción a la programación funcional

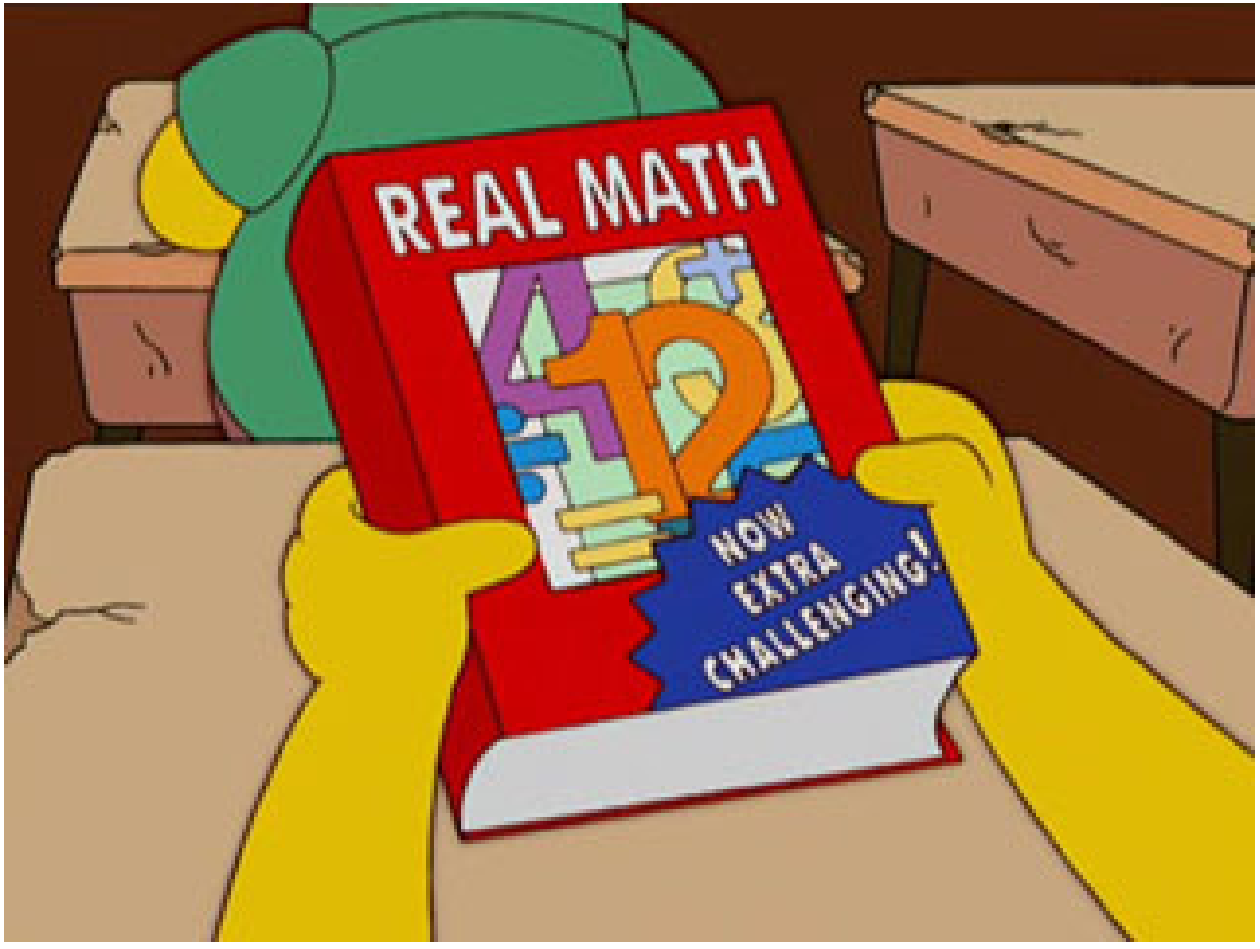
Taller de Álgebra I

Segundo cuatrimestre de 2013

Programación funcional

- Un programa en un lenguage funcional es un conjunto de ecuaciones que definen una o más funciones.
- La ejecución de un programa en este caso corresponde a la evaluación de una expresión, habitualmente solicitada desde la consola del entorno de programación.
    1. La expresión se evalúa usando las ecuaciones definidas en el programa, hasta llegar a un resultado.
    2. Las ecuaciones orientadas junto con el mecanismo de reducción describen algoritmos (definición de los pasos para resolver un problema).

Programación funcional



Programación funcional

Ejemplos de definición de funciones:

- ```
  doble ::  Int -> Int
  doble x = x + x
  ```
  ```

Programación funcional

Ejemplos de definición de funciones:

- ```
  doble ::  Int -> Int
  doble x = x + x
  ```
- ```
  fst ::  (a,b) -> a
  fst (x,y) = x
  ```

Programación funcional

Ejemplos de definición de funciones:

- ```
  doble ::  Int -> Int
  doble x = x + x
  ```
- ```
  fst ::  (a,b) -> a
  fst (x,y) = x
  ```
- ```
  dist ::  (Float,Float) -> Float
  dist (x,y) = sqrt (x^2+y^2)
  ```
  ```

Programación funcional

Más ejemplos:

- ```
  doble ::  Int -> Int
  doble x = x + x
  ```
  ```

Programación funcional

Más ejemplos:

- ```
  doble ::  Int -> Int
  doble x = x + x
  ```
- ```
  fst ::  (a,b) -> a
  fst (x,y) = x
  ```

Programación funcional

Más ejemplos:

- ```
  doble ::  Int -> Int
  doble x = x + x
  ```
- ```
  fst ::  (a,b) -> a
  fst (x,y) = x
  ```
- ```
  dist ::  (Float,Float) -> Float
  dist (x,y) = sqrt (x^2+y^2)
  ```
  ```

Programación funcional

Más ejemplos:

- ```
  doble ::  Int -> Int
  doble x = x + x
  ```
- ```
  fst ::  (a,b) -> a
  fst (x,y) = x
  ```
- ```
  dist ::  (Float,Float) -> Float
  dist (x,y) = sqrt (x^2+y^2)
  ```
- ```
  signo::  Int -> Int
  ```

Programación funcional

Más ejemplos:

- ```
  doble ::  Int -> Int
  doble x = x + x
  ```
- ```
  fst ::  (a,b) -> a
  fst (x,y) = x
  ```
- ```
  dist ::  (Float,Float) -> Float
  dist (x,y) = sqrt (x^2+y^2)
  ```
- ```
  signo::  Int -> Int
  signo 0 = 0
  signo x | x > 0 = 1
  signo x | x < 0 = -1
  ```

Tipos de datos

- Es importante observar la signatura de las funciones en las definiciones anteriores.
- Especificamos explícitamente el tipo de datos del dominio y el codominio de las funciones que definimos.

Tipos de datos

- Es importante observar la signatura de las funciones en las definiciones anteriores.
- Especificamos explícitamente el tipo de datos del dominio y el codominio de las funciones que definimos.

1. No es estrictamente necesario especificarlo, dado que el mecanismo de inferencia de tipos de Haskell puede deducir la signatura más general para cada función.
2. Sin embargo, es buena idea dar explícitamente la signatura de las funciones (¿por qué?).

Tipos de datos

- Es importante observar la signatura de las funciones en las definiciones anteriores.
- Especificamos explícitamente el tipo de datos del dominio y el codominio de las funciones que definimos.
    1. No es estrictamente necesario especificarlo, dado que el mecanismo de inferencia de tipos de Haskell puede deducir la signatura más general para cada función.
    2. Sin embargo, es buena idea dar explícitamente la signatura de las funciones (¿por qué?).
- Los identificadores `Int` y `Float` en las signaturas de los ejemplos anteriores son tipos de datos primitivos de Haskell.

Tipos de datos

- **Tipo de datos:** Conjunto de valores (llamado el conjunto base del tipo) junto con una serie de funciones que involucran al conjunto base.

[1] para nosotros lo son, para ustedes lo serán en un futuro muy cercano

Tipos de datos

- **Tipo de datos:** Conjunto de valores (llamado el conjunto base del tipo) junto con una serie de funciones que involucran al conjunto base.
- Ejemplos:
  1. $\texttt{Integer} = (\mathbb{Z}, \{+, -, *, \texttt{div}, \texttt{mod}\})$ es el tipo de datos que representa a los enteros con las operaciones aritméticas habituales.

[1] para nosotros lo son, para ustedes lo serán en un futuro muy cercano

Tipos de datos

- **Tipo de datos:** Conjunto de valores (llamado el conjunto base del tipo) junto con una serie de funciones que involucran al conjunto base.
- Ejemplos:
  1. $\texttt{Integer} = (\mathbb{Z}, \{+, -, *, \texttt{div}, \texttt{mod}\})$ es el tipo de datos que representa a los enteros con las operaciones aritméticas habituales.
  2. $\texttt{Int} = ([-2^{31}, 2^{31} - 1], \{+, -, *, \texttt{div}, \texttt{mod}\})$ es el tipo de datos que representa a los enteros representados como secuencias de 32 bits.

[1] para nosotros lo son, para ustedes lo serán en un futuro muy cercano

Tipos de datos

- **Tipo de datos:** Conjunto de valores (llamado el conjunto base del tipo) junto con una serie de funciones que involucran al conjunto base.
- Ejemplos:

  1. $\texttt{Integer} = (\mathbb{Z}, \{+, -, *, \texttt{div}, \texttt{mod}\})$ es el tipo de datos que representa a los enteros con las operaciones aritméticas habituales.
  2. $\texttt{Int} = ([-2^{31}, 2^{31} - 1], \{+, -, *, \texttt{div}, \texttt{mod}\})$ es el tipo de datos que representa a los enteros representados como secuencias de 32 bits.
  3. $\texttt{Float} = (\mathbb{Q}*, \{+, -, *, /\})$ es el tipo de datos que representa a los racionales, representados en aritmética de punto flotante.

[1] para nosotros lo son, para ustedes lo serán en un futuro muy cercano

Tipos de datos

- **Tipo de datos:** Conjunto de valores (llamado el conjunto base del tipo) junto con una serie de funciones que involucran al conjunto base.
- Ejemplos:

  1. $\texttt{Integer} = (\mathbb{Z}, \{+, -, *, \texttt{div}, \texttt{mod}\})$ es el tipo de datos que representa a los enteros con las operaciones aritméticas habituales.
  2. $\texttt{Int} = ([-2^{31}, 2^{31} - 1], \{+, -, *, \texttt{div}, \texttt{mod}\})$ es el tipo de datos que representa a los enteros representados como secuencias de 32 bits.
  3. $\texttt{Float} = (\mathbb{Q}*, \{+, -, *, /\})$ es el tipo de datos que representa a los racionales, representados en aritmética de punto flotante.
  4. $\texttt{Bool} = (\{\texttt{True}, \texttt{False}\}, \{\&\&, ||, \texttt{not}\})$ representa a los valores lógicos y las operaciones *habituales*[1].

[1] para nosotros lo son, para ustedes lo serán en un futuro muy cercano

Tipos de datos

- **Tipo de datos:** Conjunto de valores (llamado el conjunto base del tipo) junto con una serie de funciones que involucran al conjunto base.
- Ejemplos:
  1. $\texttt{Integer} = (\mathbb{Z}, \{+, -, *, \texttt{div}, \texttt{mod}\})$ es el tipo de datos que representa a los enteros con las operaciones aritméticas habituales.
  2. $\texttt{Int} = ([-2^{31}, 2^{31} - 1], \{+, -, *, \texttt{div}, \texttt{mod}\})$ es el tipo de datos que representa a los enteros representados como secuencias de 32 bits.
  3. $\texttt{Float} = (\mathbb{Q}*, \{+, -, *, /\})$ es el tipo de datos que representa a los racionales, representados en aritmética de punto flotante.
  4. $\texttt{Bool} = (\{\texttt{True}, \texttt{False}\}, \{\&\&, ||, \texttt{not}\})$ representa a los valores lógicos y las operaciones *habituales*[1].
- Dado un valor de un tipo de datos, solamente se pueden aplicar a ese valor las operaciones definidas para ese tipo de datos.

[1] para nosotros lo son, para ustedes lo serán en un futuro muy cercano

Tipos de datos

- Dados dos tipos de datos `A` y `B`, tenemos el tipo de datos `(A,B)` que representa pares ordenados de elementos, donde el primero es de tipo `A` y el segundo es de tipo `B`.

Tipos de datos

- Dados dos tipos de datos `A` y `B`, tenemos el tipo de datos `(A,B)` que representa pares ordenados de elementos, donde el primero es de tipo `A` y el segundo es de tipo `B`.
- Por ejemplo, en la siguiente definición, el dominio es un par ordenado de `Floats`:
- ```
  dist ::  (Float,Float) -> Float
  dist (x,y) = sqrt (x^2+y^2)
  ```
  ```

Tipos de datos

- Dados dos tipos de datos `A` y `B`, tenemos el tipo de datos `(A,B)` que representa pares ordenados de elementos, donde el primero es de tipo `A` y el segundo es de tipo `B`.
- Por ejemplo, en la siguiente definición, el dominio es un par ordenado de `Floats`:
- ```
  dist ::  (Float,Float) -> Float
  dist (x,y) = sqrt (x^2+y^2)
  ```
- Vamos a ver en la próxima clase un mecanismo alternativo para especificar funciones que toman más de un parámetro.
  ```

Tipos de datos: algunas preguntas

- ¿Todo lenguaje de programación tiene la noción de tipos de datos?

Tipos de datos: algunas preguntas

- ¿Todo lenguaje de programación tiene la noción de tipos de datos?
  1. Lenguajes fuertemente tipados.
  2. Lenguajes débilmente tipados.
  3. Lenguajes sin tipos de datos.

Tipos de datos: algunas preguntas

- ¿Todo lenguaje de programación tiene la noción de tipos de datos?
  1. Lenguajes fuertemente tipados.
  2. Lenguajes débilmente tipados.
  3. Lenguajes sin tipos de datos.
- ¿Por qué tenemos tipos de datos en un lenguaje?

Tipos de datos: algunas preguntas

- ¿Todo lenguaje de programación tiene la noción de tipos de datos?
  1. Lenguajes fuertemente tipados.
  2. Lenguajes débilmente tipados.
  3. Lenguajes sin tipos de datos.
- ¿Por qué tenemos tipos de datos en un lenguaje?
  1. La semántica queda explícitamente especificada.
  2. Ayuda a prevenir errores de programación.

Definiciones de funciones por casos

- Podemos usar guardas para definir funciones por casos:

- ```
  abs::  Int -> Int
  abs x | x >= 0 = x
  abs x | x < 0 = -x
  ```
  ```

Definiciones de funciones por casos

- Podemos usar guardas para definir funciones por casos:
- ```
  abs::  Int -> Int
  abs x | x >= 0 = x
  abs x | x < 0 = -x
  ```
- También podemos usar la construcción `if-then-else`, del siguiente modo:
- ```
  abs::  Int -> Int
  abs x = if x >= 0 then x else -x;
  ```

Definiciones de funciones por casos

- Podemos usar guardas para definir funciones por casos:
- ```
  abs::  Int -> Int
  abs x | x >= 0 = x
  abs x | x < 0 = -x
  ```
- También podemos usar la construcción `if-then-else`, del siguiente modo:
- ```
  abs::  Int -> Int
  abs x = if x >= 0 then x else -x;
  ```
- ¿Hay alguna diferencia entre las dos definiciones?

El tipo Bool

- El tipo de datos `Bool` representa valores de verdad, y se suele usar como valor de retorno de funciones que toman una decisión binaria.

[2] Los primeros cuatro son: 6, 28, 496 y 8128

[3] Por ejemplo 220 y 284

El tipo Bool

- El tipo de datos `Bool` representa valores de verdad, y se suele usar como valor de retorno de funciones que toman una decisión binaria.
- Ejemplos:
  1. Decidir si un número es positivo.

[2] Los primeros cuatro son: 6, 28, 496 y 8128

[3] Por ejemplo 220 y 284

El tipo Bool

- El tipo de datos `Bool` representa valores de verdad, y se suele usar como valor de retorno de funciones que toman una decisión binaria.
- Ejemplos:
  1. Decidir si un número es positivo.
  2. Decidir si un número es par.

[2] Los primeros cuatro son: 6, 28, 496 y 8128

[3] Por ejemplo 220 y 284

El tipo Bool

- El tipo de datos `Bool` representa valores de verdad, y se suele usar como valor de retorno de funciones que toman una decisión binaria.
- Ejemplos:
  1. Decidir si un número es positivo.
  2. Decidir si un número es par.
  3. Decidir si un número es primo (aja).

[2] Los primeros cuatro son: 6, 28, 496 y 8128

[3] Por ejemplo 220 y 284

El tipo Bool

- El tipo de datos `Bool` representa valores de verdad, y se suele usar como valor de retorno de funciones que toman una decisión binaria.
- Ejemplos:
  1. Decidir si un número es positivo.
  2. Decidir si un número es par.
  3. Decidir si un número es primo (aja).
  4. Decidir si un número es perfecto (epa!)[2].

[2] Los primeros cuatro son: 6, 28, 496 y 8128

[3] Por ejemplo 220 y 284

El tipo Bool

- El tipo de datos `Bool` representa valores de verdad, y se suele usar como valor de retorno de funciones que toman una decisión binaria.
- Ejemplos:
  1. Decidir si un número es positivo.
  2. Decidir si un número es par.
  3. Decidir si un número es primo (aja).
  4. Decidir si un número es perfecto (epa!)[2].
  5. Decidir si dos números son amigos (wtf?)[3].

[2] Los primeros cuatro son: 6, 28, 496 y 8128

[3] Por ejemplo 220 y 284

El tipo Bool

- Los conectivos lógicos se definen por medio de ecuaciones, al igual que las funciones que vimos en los ejemplos anteriores:

- ```
  y ::  (Bool, Bool) -> Bool
  y (True, False) = False
  y (True, True) = True
  y (False, False) = False
  y (False, True) = False
  ```
  ```

El tipo Bool

- Los conectivos lógicos se definen por medio de ecuaciones, al igual que las funciones que vimos en los ejemplos anteriores:

- ```
  y ::  (Bool, Bool) -> Bool
  y (True, False) = False
  y (True, True) = True
  y (False, False) = False
  y (False, True) = False
  ```

- Una definición alternativa:

- ```
  y ::  (Bool, Bool) -> Bool
  y (True, x) = x
  y (False, x) = False
  ```

El tipo Bool

- Los conectivos lógicos se definen por medio de ecuaciones, al igual que las funciones que vimos en los ejemplos anteriores:

- ```
  y ::  (Bool, Bool) -> Bool
  y (True, False) = False
  y (True, True) = True
  y (False, False) = False
  y (False, True) = False
  ```

- Una definición alternativa:

- ```
  y ::  (Bool, Bool) -> Bool
  y (True, x) = x
  y (False, x) = False
  ```

- En la última ecuación, la `x` se puede reemplazar por un `_` (guión bajo).

El tipo Bool

- Los conectivos lógicos se definen por medio de ecuaciones, al igual que las funciones que vimos en los ejemplos anteriores:

- ```
  y ::  (Bool, Bool) -> Bool
  y (True, False) = False
  y (True, True) = True
  y (False, False) = False
  y (False, True) = False
  ```

- Una definición alternativa:

- ```
  y ::  (Bool, Bool) -> Bool
  y (True, x) = x
  y (False, x) = False
  ```

- En la última ecuación, la `x` se puede reemplazar por un `_` (guión bajo).
- ¿Cómo se definen los otros conectivos?

Ejercicios

1. Programar la siguiente función:

$$f(n) = \begin{cases} n & \text{si } n < 10 \\ n+1 & \text{si } n \geq 10 \end{cases}$$

2. Programar la función $f(n) = |n|$ usando guardas y usando la construcción **if-then-else**.
3. Programar la función $\text{nand}(x, y) = \neg(x \wedge y)$, usando funciones predefinidas de Haskell y también por medio de su tabla de verdad.
4. Repetir el ejercicio anterior para la función $\text{nor}(x, y) = \neg(x \vee y)$.
5. Programar una función que tome tres parámetros $a, b, c \in \mathbb{R}$ y que calcule las raíces de la función cuadrática $f(x) = ax^2 + bx + c$.