

Tema 3: Tipos y clases

Programación declarativa (2009–10)

José A. Alonso Jiménez

Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

IM Tema 3: Tipos y clases

Tema 3: Tipos y clases

1. Conceptos básicos sobre tipos
2. Tipos básicos
3. Tipos compuestos
 - Tipos listas
 - Tipos tuplas
 - Tipos funciones
4. Parcialización
5. Polimorfismo y sobrecarga
 - Tipos polimórficos
 - Tipos sobrecargados
6. Clases básicas

Tema 3: Tipos y clases

1. Conceptos básicos sobre tipos
2. Tipos básicos
3. Tipos compuestos
4. Parcialización
5. Polimorfismo y sobrecarga
6. Clases básicas

¿Qué es un tipo?

- ▶ Un **tipo** es una colección de valores relacionados.
- ▶ Un ejemplo de tipos es el de los valores booleanos: `Bool`
- ▶ El tipo `Bool` tiene dos valores `True` (verdadero) y `False` (falso).
- ▶ $v :: T$ representa que v es un valor del tipo T y se dice que “ v tiene tipo T ”.
- ▶ Cálculo de tipo con `:type`

```
Prelude> :type True
True :: Bool
Prelude> :type False
False :: Bool
```
- ▶ El tipo `Bool -> Bool` está formado por todas las funciones cuyo argumento y valor son booleanos.
- ▶ Ejemplo de tipo `Bool -> Bool`

```
Prelude> :type not
not :: Bool -> Bool
```

Inferencia de tipos

- ▶ Regla de inferencia de tipos

$$\frac{f :: A \rightarrow B \quad e :: A}{f e :: B}$$

- ▶ Tipos de expresiones:

```
Prelude> :type not True
```

```
not True :: Bool
```

```
Prelude> :type not False
```

```
not False :: Bool
```

```
Prelude> :type not (not False)
```

```
not (not False) :: Bool
```

- ▶ Error de tipo:

```
Prelude> :type not 3
```

```
Error: No instance for (Num Bool)
```

```
Prelude> :type 1 + False
```

```
Error: No instance for (Num Bool)
```

5 / 33

Ventajas de los tipos

- ▶ Los lenguajes en los que la inferencia de tipo precede a la evaluación se denominan de **tipos seguros**.
- ▶ Haskell es un lenguaje de tipos seguros.
- ▶ En los lenguajes de tipos seguros no ocurren errores de tipos durante la evaluación.
- ▶ La inferencia de tipos no elimina todos los errores durante la evaluación. Por ejemplo,

```
Prelude> :type 1 'div' 0
1 'div' 0 :: (Integral t) => t
Prelude> 1 'div' 0
*** Exception: divide by zero
```

6 / 33

Tema 3: Tipos y clases

1. Conceptos básicos sobre tipos
2. Tipos básicos
3. Tipos compuestos
4. Parcialización
5. Polimorfismo y sobrecarga
6. Clases básicas

Tipos básicos

- ▶ Bool (**Valores lógicos**):
 - ▶ Sus valores son True y False.
- ▶ Char (**Caracteres**):
 - ▶ Ejemplos: 'a', 'B', '3', '+'
- ▶ String (**Cadena de caracteres**):
 - ▶ Ejemplos: "abc", "1 + 2 = 3"
- ▶ Int (**Enteros de precisión fija**):
 - ▶ Enteros entre -2^{31} y $2^{31} - 1$.
 - ▶ Ejemplos: 123, -12
- ▶ Integer (**Enteros de precisión arbitraria**):
 - ▶ Ejemplos: 1267650600228229401496703205376.
- ▶ Float (**Reales de precisión arbitraria**):
 - ▶ Ejemplos: 1.2, -23.45, 45e-7
- ▶ Double (**Reales de precisión doble**):
 - ▶ Ejemplos: 1.2, -23.45, 45e-7

Tema 3: Tipos y clases

1. Conceptos básicos sobre tipos

2. Tipos básicos

3. Tipos compuestos

Tipos listas

Tipos tuplas

Tipos funciones

4. Parcialización

5. Polimorfismo y sobrecarga

Tipos listas

▶ Una **lista** es una sucesión de elementos del mismo tipo.

▶ `[T]` es el tipo de las listas de elementos de tipo `T`.

▶ Ejemplos de listas:

```
[False, True]  :: [Bool]
['a', 'b', 'd'] :: [Char]
["uno", "tres"] :: [String]
```

▶ Longitudes:

▶ La **longitud** de una lista es el número de elementos.

▶ La lista de longitud 0, `[]`, es la **lista vacía**.

▶ Las listas de longitud 1 se llaman **listas unitarias**.

▶ Comentarios:

▶ El tipo de una lista no informa sobre su longitud:

```
['a', 'b']  :: [Char]
['a', 'b', 'c'] :: [Char]
```

▶ El tipo de los elementos de una lista puede ser cualquiera:

```
[[ 'a', 'b' ], [ 'c' ]] :: [[Char]]
```

Tipos tuplas

- ▶ Una **tupla** es una sucesión de elementos.
- ▶ (T_1, T_2, \dots, T_n) es el tipo de las n -tuplas cuya componente i -ésima es de tipo T_i .

- ▶ Ejemplos de tuplas:

```
(False, True)      :: (Bool, Bool)
(False, 'a', True) :: (Bool, Char, Bool)
```

- ▶ Aridades:

- ▶ La **aridad** de una tupla es el número de componentes.
- ▶ La tupla de aridad 0, $()$, es la **tupla vacía**.
- ▶ No están permitidas las tuplas de longitud 1.

- ▶ Comentarios:

- ▶ El tipo de una tupla informa sobre su longitud:

```
('a', 'b')      :: (Char, Char)
('a', 'b', 'c') :: (Char, Char, Char)
```

- ▶ El tipo de los elementos de una tupla puede ser cualquiera:

```
(('a', 'b'), ['c', 'd']) :: ((Char, Char), [Char])
```

Tipos funciones

- ▶ Una **función** es una aplicación de valores de un tipo en valores de otro tipo.
- ▶ $T_1 \rightarrow T_2$ es el tipo de las funciones que aplica valores del tipo T_1 en valores del tipo T_2 .
- ▶ Ejemplos de funciones:

```
not      :: Bool -> Bool
isDigit  :: Char -> Bool
```

Funciones con múltiples argumentos o valores

- ▶ Ejemplo de función con múltiples argumentos:
suma (x,y) es la suma de x e y. Por ejemplo, suma (2,3) es 5.

```
suma :: (Int,Int) -> Int
```

```
suma (x,y) = x+y
```

- ▶ Ejemplo de función con múltiples valores:
deCeroA n es la lista de los números desde 0 hasta n. Por ejemplo, deCeroA n es [0,1,2,3,4,5].

```
deCeroA :: Int -> [Int]
```

```
deCeroA n = [0..n]
```

- ▶ Notas:
 1. En las definiciones se ha escrito la **signatura** de las funciones.
 2. No es obligatorio escribir la signatura de las funciones.
 3. Es conveniente escribir las signatura.

Tema 3: Tipos y clases

1. Conceptos básicos sobre tipos
2. Tipos básicos
3. Tipos compuestos
4. **Parcialización**
5. Polimorfismo y sobrecarga
6. Clases básicas

Parcialización

- ▶ Mecanismo de **parcialización** (*currying* en inglés): Las funciones de más de un argumento pueden interpretarse como funciones que toman un argumento y devuelven otra función con un argumento menos.
- ▶ Ejemplo de parcialización:

```
suma' :: Int -> (Int -> Int)
suma' x y = x+y
```

suma' toma un entero x y devuelve la función suma' x que toma un entero y y devuelve la suma de x e y. Por ejemplo,

```
*Main> :type suma' 2
suma' 2 :: Int -> Int
*Main> :type suma' 2 3
suma' 2 3 :: Int
```

Parcialización con tres argumentos

- ▶ Ejemplo de parcialización con tres argumentos:

```
mult :: Int -> (Int -> (Int -> Int))
mult x y z = x*y*z
```

mult toma un entero x y devuelve la función mult x que toma un entero y y devuelve la función mult x y que toma un entero z y devuelve x*y*z. Por ejemplo,

```
*Main> :type mult 2
mult 2 :: Int -> (Int -> Int)
*Main> :type mult 2 3
mult 2 3 :: Int -> Int
*Main> :type mult 2 3 7
mult 2 3 7 :: Int
```

Aplicación parcial

- ▶ Las funciones que toman sus argumentos de uno en uno se llaman **currificadas** (*curried* en inglés).
- ▶ Las funciones `suma'` y `mult` son currificadas.
- ▶ Las funciones currificadas pueden aplicarse parcialmente. Por ejemplo,

```
*Main> (suma' 2) 3
5
```

- ▶ Pueden definirse funciones usando aplicaciones parciales. Por ejemplo,

```
suc :: Int -> Int
suc = suma' 1
```

`suc x` es el sucesor de `x`. Por ejemplo, `suc 2` es 3.

Convenios para reducir paréntesis

- ▶ Convenio 1: Las flechas en los tipos se asocia por la derecha. Por ejemplo,

```
Int -> Int -> Int -> Int
```

representa a

```
Int -> (Int -> (Int -> Int))
```

- ▶ Convenio 2: Las aplicaciones de funciones se asocia por la izquierda. Por ejemplo,

```
mult x y z
```

representa a

```
((mult x) y) z
```

- ▶ **Nota:** Todas las funciones con múltiples argumentos se definen en forma currificada, salvo que explícitamente se diga que los argumentos tienen que ser tuplas.

Tema 3: Tipos y clases

1. Conceptos básicos sobre tipos

2. Tipos básicos

3. Tipos compuestos

4. Parcialización

5. Polimorfismo y sobrecarga

Tipos polimórficos

Tipos sobrecargados

6. Clases básicas

19 / 33

Tipos polimórficos

- ▶ Un tipo es **polimórfico** (“tiene muchas formas”) si contiene una variable de tipo.
- ▶ Una función es **polimórfica** si su tipo es polimórfico.
- ▶ La función `length` es polimórfica:
 - ▶ Comprobación:

```
| Prelude> :type length  
| length :: [a] -> Int
```

- ▶ Significa que para cualquier tipo `a`, `length` toma una lista de elementos de tipo `a` y devuelve un entero.
- ▶ `a` es una variable de tipos.
- ▶ Las variables de tipos tienen que empezar por minúscula.
- ▶ Ejemplos:

```
| length [1, 4, 7, 1]           ~> 4  
| length ["Lunes", "Martes", "Jueves"] ~> 3  
| length [reverse, tail]      ~> 2
```

20 / 33

Ejemplos de funciones polimórficas

- ▶ `fst :: (a, b) -> a`
 - | `fst (1, 'x')` \rightsquigarrow 1
 - | `fst (True, "Hoy")` \rightsquigarrow True
- ▶ `head :: [a] -> a`
 - | `head [2,1,4]` \rightsquigarrow 2
 - | `head ['b', 'c']` \rightsquigarrow 'b'
- ▶ `take :: Int -> [a] -> [a]`
 - | `take 3 [3,5,7,9,4]` \rightsquigarrow [3,5,7]
 - | `take 2 ['1', 'o', '1', 'a']` \rightsquigarrow "lo"
 - | `take 2 "lola"` \rightsquigarrow "lo"
- ▶ `zip :: [a] -> [b] -> [(a, b)]`
 - | `zip [3,5] "lo"` \rightsquigarrow [(3, 'l'), (5, 'o')]
- ▶ `id :: a -> a`
 - | `id 3` \rightsquigarrow 3
 - | `id 'x'` \rightsquigarrow 'x'

Tipos sobrecargados

- ▶ Un tipo está **sobrecargado** si contiene una restricción de clases.
- ▶ Una función está **sobrecargada** si su tipo está sobrecargado.
- ▶ La función `sum` está sobrecargada:

- ▶ Comprobación:

```
Prelude> :type sum
sum :: (Num a) => [a] -> a
```

- ▶ Significa que para cualquier tipo numérico `a`, `sum` toma una lista de elementos de tipo `a` y devuelve un valor de tipo `a`.
- ▶ `Num a` es una restricción de clases.
- ▶ Las restricciones de clases son expresiones de la forma `C a`, donde `C` es el nombre de una clase y `a` es una variable de tipo.
- ▶ Ejemplos:

```
sum [2, 3, 5]                       $\rightsquigarrow$  10
sum [2.1, 3.23, 5.345]            $\rightsquigarrow$  10.675
```

Ejemplos de tipos sobrecargados

- ▶ Ejemplos de funciones sobrecargadas:
 - ▶ `(-)` :: (Num a) => a -> a -> a
 - ▶ `(*)` :: (Num a) => a -> a -> a
 - ▶ `negate` :: (Num a) => a -> a
 - ▶ `abs` :: (Num a) => a -> a
 - ▶ `signum` :: (Num a) => a -> a
- ▶ Ejemplos de números sobrecargados:
 - ▶ `5` :: (Num t) => t
 - ▶ `5.2` :: (Fractional t) => t

Tema 3: Tipos y clases

1. Conceptos básicos sobre tipos
2. Tipos básicos
3. Tipos compuestos
4. Parcialización
5. Polimorfismo y sobrecarga
6. Clases básicas

Clases básicas

- ▶ Una **clase** es una colección de tipos junto con ciertas operaciones sobrecargadas llamadas **métodos**.
- ▶ Clases básicas:

Eq	tipos comparables por igualdad
Ord	tipos ordenados
Show	tipos mostrables
Read	tipos legibles
Num	tipos numéricos
Integral	tipos enteros
Fractional	tipos fraccionarios

La clase Eq (tipos comparables por igualdad)

- ▶ Eq contiene los tipos cuyos valores son comparables por igualdad.
- ▶ Métodos:
 - `(==) :: a -> a -> Bool`
 - `(/=) :: a -> a -> Bool`
- ▶ Instancias:
 - ▶ Bool, Char, String, Int, Integer, Float y Double.
 - ▶ tipos compuestos: listas y tuplas.
- ▶ Ejemplos:

```
False == True      ~> False
False /= True      ~> True
'a' == 'b'         ~> False
"aei" == "aei"     ~> True
[2,3] == [2,3,2]   ~> False
('a',5) == ('a',5) ~> True
```

La clase Ord (tipos ordenados)

- ▶ Ord es la subclase de Eq de tipos cuyos valores están ordenados.

- ▶ Métodos:

```
(<), (<=), (>), (>=) :: a -> a -> Bool
min, max           :: a -> a -> a
```

- ▶ Instancias:

- ▶ Bool, Char, String, Int, Integer, Float y Double.
- ▶ tipos compuestos: listas y tuplas.

- ▶ Ejemplos:

```
False < True           ~> True
min 'a' 'b'           ~> 'a'
"elegante" < "elefante" ~> False
[1,2,3] < [1,2]       ~> False
('a',2) < ('a',1)     ~> False
('a',2) < ('b',1)     ~> True
```

La clase Show (tipos mostrables)

- ▶ Show contiene los tipos cuyos valores se pueden convertir en cadenas de caracteres.

- ▶ Método:

```
show :: a -> String
```

- ▶ Instancias:

- ▶ Bool, Char, String, Int, Integer, Float y Double.
- ▶ tipos compuestos: listas y tuplas.

- ▶ Ejemplos:

```
show False           ~> "False"
show 'a'             ~> "'a'"
show 123              ~> "123"
show [1,2,3]         ~> "[1,2,3]"
show ('a',True)      ~> "('a',True)"
```

La clase Read (tipos legibles)

- ▶ Read contiene los tipos cuyos valores se pueden obtener a partir de cadenas de caracteres.

- ▶ Método:

```
| read :: String -> a
```

- ▶ Instancias:

- ▶ Bool, Char, String, Int, Integer, Float y Double.
- ▶ tipos compuestos: listas y tuplas.

- ▶ Ejemplos:

```
| read "False" :: Bool           ~> False
| read "'a'"  :: Char           ~> 'a'
| read "123"  :: Int            ~> 123
| read "[1,2,3]" :: [Int]       ~> [1,2,3]
| read "('a',True)" :: (Char,Bool) ~> ('a',True)
```

La clase Num (tipos numéricos)

- ▶ Num es la subclase de Eq y Ord de tipos cuyos valores son números

- ▶ Métodos:

```
| (+), (*), (-)      :: a -> a -> a
| negate, abs, signum :: a -> a
```

- ▶ Instancias: Int, Integer, Float y Double.

- ▶ Ejemplos:

```
| 2+3           ~> 5
| 2.3+4.2       ~> 6.5
| negate 2.7    ~> -2.7
| abs (-5)     ~> 5
| signum (-5)  ~> -1
```

La clase `Integral` (tipos enteros)

- ▶ `Integral` es la subclase de `Num` cuyo tipos tienen valores enteros.

- ▶ Métodos:

```
| div :: a -> a -> a  
| mod :: a -> a -> a
```

- ▶ Instancias: `Int` e `Integer`.

- ▶ Ejemplos:

```
| 11 'div' 4 ~> 2  
| 11 'mod' 4 ~> 3
```

La clase `Fractional` (tipos fraccionarios)

- ▶ `Fractional` es la subclase de `Num` cuyo tipos tienen valores no son enteros.

- ▶ Métodos:

```
| (/) :: a -> a -> a  
| recip :: a -> a
```

- ▶ Instancias: `Float` y `Double`.

- ▶ Ejemplos:

```
| 7.0 / 2.0 ~> 3.5  
| recip 0.2 ~> 5.0
```

Bibliografía

1. R. Bird. *Introducción a la programación funcional con Haskell*. Prentice Hall, 2000.
 - ▶ Cap. 2: Tipos de datos simples.
2. G. Hutton *Programming in Haskell*. Cambridge University Press, 2007.
 - ▶ Cap. 3: Types and classes.
3. B. O'Sullivan, D. Stewart y J. Goerzen *Real World Haskell*. O'Reilly, 2008.
 - ▶ Cap. 2: Types and Functions.
4. B.C. Ruiz, F. Gutiérrez, P. Guerrero y J.E. Gallardo. *Razonando con Haskell*. Thompson, 2004.
 - ▶ Cap. 2: Introducción a Haskell.
 - ▶ Cap. 5: El sistema de clases de Haskell.
5. S. Thompson. *Haskell: The Craft of Functional Programming*, Second Edition. Addison-Wesley, 1999.
 - ▶ Cap. 3: Basic types and definitions.