

# Programación Funcional

## Introducción a los tipos de datos en Haskell

Mariano Rean

Universidad Nacional de Hurlingham

18 de Abril, 2020

## Tipos de datos

En GHCi podemos ver el tipo de un valor y la signatura de una función usando `:t`.

# Tipos de datos

En GHCi podemos ver el tipo de un valor y la signatura de una función usando `:t`.

```
1 *Main> :t True
2 True  :: Bool
```

# Tipos de datos

En GHCi podemos ver el tipo de un valor y la signatura de una función usando `:t`.

```
1 *Main> :t True
2 True  :: Bool
```

```
1 *Main> :t 3^48 < 2^23
2 3^48 < 2^23 :: Bool
```

# Tipos de datos

En GHCi podemos ver el tipo de un valor y la signatura de una función usando `:t`.

```
1 *Main> :t True
2 True  :: Bool
```

```
1 *Main> :t 3^48 < 2^23
2 3^48 < 2^23 :: Bool
```

```
1 maximo :: Int -> Int -> Int
2 maximo x y | x >= y      = x
3             | otherwise = y
```

# Tipos de datos

En GHCi podemos ver el tipo de un valor y la signatura de una función usando `:t`.

```
1 *Main> :t True
2 True  :: Bool
```

```
1 *Main> :t 3^48 < 2^23
2 3^48 < 2^23 :: Bool
```

```
1 maximo :: Int -> Int -> Int
2 maximo x y | x >= y      = x
3             | otherwise = y
```

```
1 *Main> :t maximo
2 maximo :: Int -> Int -> Int
```

# Tipos de datos

En GHCi podemos ver el tipo de un valor y la signatura de una función usando `:t`.

```
1 *Main> :t True
2 True :: Bool
```

```
1 *Main> :t 3^48 < 2^23
2 3^48 < 2^23 :: Bool
```

```
1 maximo :: Int -> Int -> Int
2 maximo x y | x >= y      = x
3             | otherwise = y
```

```
1 *Main> :t maximo
2 maximo :: Int -> Int -> Int
```

```
1 *Main> :t maximo 5 6
2 maximo 5 6 :: Int
```

# Clases de tipos

## Clase de tipos

Un **conjunto de tipos de datos** a los que se les puede aplicar un **conjunto de funciones**.



# Clases de tipos

## Clase de tipos

Un **conjunto de tipos de datos** a los que se les puede aplicar un **conjunto de funciones**.

## Algunas clases:

1.  $\text{Integral} := (\{ \text{Int}, \text{Integer}, \dots \}, \{ \text{mod}, \text{div}, \dots \})$

# Clases de tipos

## Clase de tipos

Un **conjunto de tipos de datos** a los que se les puede aplicar un **conjunto de funciones**.

## Algunas clases:

1.  $\text{Integral} := (\{ \text{Int}, \text{Integer}, \dots \}, \{ \text{mod}, \text{div}, \dots \})$
2.  $\text{Fractional} := (\{ \text{Float}, \text{Double}, \dots \}, \{ (/), \dots \})$

# Clases de tipos

## Clase de tipos

Un **conjunto de tipos de datos** a los que se les puede aplicar un **conjunto de funciones**.

## Algunas clases:

1.  $\text{Integral} := (\{ \text{Int}, \text{Integer}, \dots \}, \{ \text{mod}, \text{div}, \dots \})$
2.  $\text{Fractional} := (\{ \text{Float}, \text{Double}, \dots \}, \{ (/), \dots \})$
3.  $\text{Floating} := (\{ \text{Float}, \text{Double}, \dots \}, \{ \text{sqrt}, \text{sin}, \text{cos}, \text{tan}, \dots \})$

# Clases de tipos

## Clase de tipos

Un **conjunto de tipos de datos** a los que se les puede aplicar un **conjunto de funciones**.

## Algunas clases:

1.  $\text{Integral} := (\{ \text{Int}, \text{Integer}, \dots \}, \{ \text{mod}, \text{div}, \dots \})$
2.  $\text{Fractional} := (\{ \text{Float}, \text{Double}, \dots \}, \{ (/), \dots \})$
3.  $\text{Floating} := (\{ \text{Float}, \text{Double}, \dots \}, \{ \text{sqrt}, \text{sin}, \text{cos}, \text{tan}, \dots \})$
4.  $\text{Num} := (\{ \text{Int}, \text{Integer}, \text{Float}, \text{Double}, \dots \}, \{ (+), (*), \text{abs}, \dots \})$

# Clases de tipos

## Clase de tipos

Un **conjunto de tipos de datos** a los que se les puede aplicar un **conjunto de funciones**.

## Algunas clases:

1.  $\text{Integral} := (\{ \text{Int}, \text{Integer}, \dots \}, \{ \text{mod}, \text{div}, \dots \})$
2.  $\text{Fractional} := (\{ \text{Float}, \text{Double}, \dots \}, \{ (/), \dots \})$
3.  $\text{Floating} := (\{ \text{Float}, \text{Double}, \dots \}, \{ \text{sqrt}, \text{sin}, \text{cos}, \text{tan}, \dots \})$
4.  $\text{Num} := (\{ \text{Int}, \text{Integer}, \text{Float}, \text{Double}, \dots \}, \{ (+), (*), \text{abs}, \dots \})$
5.  $\text{Ord} := (\{ \text{Bool}, \text{Int}, \text{Integer}, \text{Float}, \text{Double}, \dots \}, \{ (<=), \text{compare} \})$

# Clases de tipos

## Clase de tipos

Un **conjunto de tipos de datos** a los que se les puede aplicar un **conjunto de funciones**.

## Algunas clases:

1.  $\text{Integral} := (\{ \text{Int}, \text{Integer}, \dots \}, \{ \text{mod}, \text{div}, \dots \})$
2.  $\text{Fractional} := (\{ \text{Float}, \text{Double}, \dots \}, \{ (/), \dots \})$
3.  $\text{Floating} := (\{ \text{Float}, \text{Double}, \dots \}, \{ \text{sqrt}, \text{sin}, \text{cos}, \text{tan}, \dots \})$
4.  $\text{Num} := (\{ \text{Int}, \text{Integer}, \text{Float}, \text{Double}, \dots \}, \{ (+), (*), \text{abs}, \dots \})$
5.  $\text{Ord} := (\{ \text{Bool}, \text{Int}, \text{Integer}, \text{Float}, \text{Double}, \dots \}, \{ (<=), \text{compare} \})$
6.  $\text{Eq} := (\{ \text{Bool}, \text{Int}, \text{Integer}, \text{Float}, \text{Double}, \dots \}, \{ (==), (/=) \})$

## Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su `signatura`. ¿Qué tipos de datos admite esta función?

## Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su signatura. ¿Qué tipos de datos admite esta función?

¿Qué pasa con...? ¿funcionan?

► `triple 2`



## Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su signatura. ¿Qué tipos de datos admite esta función?

¿Qué pasa con...? ¿funcionan?

► `triple 2 ==> 6`

## Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su signatura. ¿Qué tipos de datos admite esta función?

¿Qué pasa con...? ¿funcionan?

▶ `triple 2`  $\rightsquigarrow$  6

▶ `triple 2.5`

## Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su signatura. ¿Qué tipos de datos admite esta función?

¿Qué pasa con...? ¿funcionan?

► `triple 2`  $\rightsquigarrow$  6

► `triple 2.5`  $\rightsquigarrow$  7.5

## Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su signatura. ¿Qué tipos de datos admite esta función?

¿Qué pasa con...? ¿funcionan?

- ▶ `triple 2`  $\rightsquigarrow$  6
- ▶ `triple 2.5`  $\rightsquigarrow$  7.5
- ▶ `triple True`

# Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su signatura. ¿Qué tipos de datos admite esta función?

¿Qué pasa con...? ¿funcionan?

▶ `triple 2`  $\rightsquigarrow$  6

▶ `triple 2.5`  $\rightsquigarrow$  7.5

▶ `triple True` **¡error de tipos!**

No instance for (Num Bool) arising from a use of ‘\*’

# Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su signatura. ¿Qué tipos de datos admite esta función?

¿Qué pasa con...? ¿funcionan?

▶ `triple 2`  $\rightsquigarrow$  6

▶ `triple 2.5`  $\rightsquigarrow$  7.5

▶ `triple True` **¡error de tipos!**

No instance for (Num Bool) arising from a use of ‘\*’

```
1 *Main> :t triple
2 triple :: Num a => a -> a
```

# Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su signatura. ¿Qué tipos de datos admite esta función?

¿Qué pasa con...? ¿funcionan?

▶ `triple 2`  $\rightsquigarrow$  6

▶ `triple 2.5`  $\rightsquigarrow$  7.5

▶ `triple True` **¡error de tipos!**

No instance for (Num Bool) arising from a use of ‘\*’

```
1 *Main> :t triple
2 triple :: Num a => a -> a
```

¿Qué significa `Num a => ...` ?

# Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su signatura. ¿Qué tipos de datos admite esta función?

¿Qué pasa con...? ¿funcionan?

▶ `triple 2`  $\rightsquigarrow$  6

▶ `triple 2.5`  $\rightsquigarrow$  7.5

▶ `triple True` **¡error de tipos!**

No instance for (Num Bool) arising from a use of ‘\*’

```
1 *Main> :t triple
2 triple :: Num a => a -> a
```

¿Qué significa `Num a => ...` ?

Acá `a` es una *variable de tipo*, y `Num a => ...` quiere decir que `a` puede ser cualquier tipo que pertenezca a la clase `Num`.



# Clases de tipos

Supongamos que definimos la función `triple x = 3*x` sin definir su signatura. ¿Qué tipos de datos admite esta función?

¿Qué pasa con...? ¿funcionan?

▶ `triple 2`  $\rightsquigarrow$  6

▶ `triple 2.5`  $\rightsquigarrow$  7.5

▶ `triple True` ¡error de tipos!

No instance for (Num Bool) arising from a use of ‘\*’

```
1 *Main> :t triple
2 triple :: Num a => a -> a
```

¿Qué significa `Num a => ...` ?

Acá `a` es una *variable de tipo*, y `Num a => ...` quiere decir que `a` puede ser cualquier tipo que pertenezca a la clase `Num`.

En resumen, la función `triple` solo admite tipos de datos numéricos.

# Funciones polimórficas

Es posible definir funciones usando **variables de tipo** para que puedan funcionar sobre muchos tipos de datos.

# Funciones polimórficas

Es posible definir funciones usando **variables de tipo** para que puedan funcionar sobre muchos tipos de datos.

Por ejemplo:

```
1 identidad :: a -> a
2 identidad x = x
```

# Funciones polimórficas

Es posible definir funciones usando **variables de tipo** para que puedan funcionar sobre muchos tipos de datos.

Por ejemplo:

```
1 identidad :: a -> a
2 identidad x = x
```

Notar que a va necesariamente en minúscula.

# Funciones polimórficas

Es posible definir funciones usando **variables de tipo** para que puedan funcionar sobre muchos tipos de datos.

Por ejemplo:

```
1 identidad :: a -> a
2 identidad x = x
```

Notar que `a` va necesariamente en minúscula.

En Haskell esa función ya existe, se llama `id` y vale para cualquier tipo de datos.

```
1 *Main> :t id
2 id :: a -> a
```

# Funciones polimórficas

Es posible definir funciones usando **variables de tipo** para que puedan funcionar sobre muchos tipos de datos.

Por ejemplo:

```
1 identidad :: a -> a
2 identidad x = x
```

Notar que `a` va necesariamente en minúscula.

En Haskell esa función ya existe, se llama `id` y vale para cualquier tipo de datos.

```
1 *Main> :t id
2 id :: a -> a
```

Podemos definir nosotros, por ejemplo:

```
1 cuadruple :: Num a => a -> a
2 cuadruple x = 4*x
```

## Nueva familia de tipos: Tuplas

- Dados dos tipos de datos A y B, podemos crear 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 (Int, Int), (Float, Int), (Bool, (Float, Int)).

## Nueva familia de tipos: Tuplas

- ▶ Dados dos tipos de datos A y B, podemos crear 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 (Int, Int), (Float, Int), (Bool, (Float, Int)).
- ▶ Algunas funciones para tuplas de dos elementos:



## Nueva familia de tipos: Tuplas

- ▶ Dados dos tipos de datos A y B, podemos crear 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 (Int, Int), (Float, Int), (Bool, (Float, Int)).
- ▶ Algunas funciones para tuplas de dos elementos:
  - ▶ `fst :: (a, b) -> a`      Ejemplo de uso: `fst (1 + 4, 2)`  
     $\rightsquigarrow 5$

## Nueva familia de tipos: Tuplas

- ▶ Dados dos tipos de datos A y B, podemos crear 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 (Int, Int), (Float, Int), (Bool, (Float, Int)).
- ▶ Algunas funciones para tuplas de dos elementos:
  - ▶ `fst :: (a, b) -> a`                      Ejemplo de uso: `fst (1 + 4, 2)`  
     $\rightsquigarrow 5$
  - ▶ `snd :: (a, b) -> b`                      Ejemplo de uso: `snd (1, (2, 3))`  
     $\rightsquigarrow (2, 3)$

## Nueva familia de tipos: Tuplas

- ▶ Dados dos tipos de datos A y B, podemos crear 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 (Int, Int), (Float, Int), (Bool, (Float, Int)).
- ▶ Algunas funciones para tuplas de dos elementos:
  - ▶ `fst :: (a, b) -> a`                      Ejemplo de uso: `fst (1 + 4, 2)`  
     $\rightsquigarrow 5$
  - ▶ `snd :: (a, b) -> b`                      Ejemplo de uso: `snd (1, (2, 3))`  
     $\rightsquigarrow (2, 3)$
- ▶ Podemos crear también ternas, cuaternas, etc...

## Ejercicios

Implementar las siguientes funciones, especificando su signatura.

1. `prodInt`: calcula el producto interno entre dos vectores de  $\mathbb{R}^2$ .
2. `todoMenor`: dados dos vectores de  $\mathbb{R}^2$ , decide si es cierto que cada coordenada del primer vector es menor a la coordenada correspondiente del segundo vector.
3. `distanciaPuntos`: calcula la distancia entre dos puntos de  $\mathbb{R}^2$ .
4. `sumaTerna`: dada una terna de enteros, calcula la suma de sus tres elementos.
5. `posicPrimerPar`: dada una terna de enteros, devuelve la posición del primer número par si es que hay alguno, y devuelve 4 si son todos impares.
6. `crearPar :: a -> b -> (a, b)`: crea un par a partir de sus dos componentes dadas por separado (debe funcionar para elementos de cualquier tipo).
7. `invertir :: (a, b) -> (b, a)`: invierte los elementos del par pasado como parámetro (debe funcionar para elementos de cualquier tipo).