# Programación Funcional

Curso 2019-20

## INTRODUCIENDO LA PROGRAMACIÓN CON HASKELL

# Lenguaje Haskell
## (Haskell B. Curry: lógico-matemático 1900-1982)

### Descarga del sistema

www.haskell.org/platform/

### Trabajamos con el intérprete

ghci

### Configura el intérprete a tu gusto

### Primeros contactos con Haskell

Trabajamos en línea de comandos

**Cómputos $\equiv$ evaluación de expresiones**

# Cómputos ≡ evaluación de expresiones (I)

```
> 29^25
363036212362725866319302825147433074 9
> div 8 3
2
> 8/3
2.6666666666666665
> True && (False || not False)
 True
 > not (not True)
 True
```

Enteros, de varios tipos

Reales, de varias precisiones

Booleanos

Constantes en mayúscula

Funciones y variables en minúscula

Operadores infijos ( `^` , `/`, `&&`, `||`)

Argumentos de las funciones sin paréntesis

☞notación *currificada* (ver `div` y `not`)

Paréntesis para anidamiento de funciones

(y para inhibir prioridades de operadores)

# Cómputos ≡ evaluación de expresiones (II)

```
> true && False
```
Not in scope:
'true'  `true` variable sin definir

Error en tiempo de compilación

```
> div 1 0
```
Exception  Expresión con valor no definido

Error en tiempo de ejecución

```
> 1 + True
```
Error de tipo  Las expresiones deben estar bien tipadas

Error en tiempo de compilación

```
> False && div 1 0 == 2
False
```
En Haskell las expresiones solo se evalúan si hace falta

☞ Evaluación perezosa (*lazy evaluation*)

```
> False && 0
```
Error de tipo  El tipado es estático

```
> False && div 1 0 == 'a'
```
Error de tipo  El tipado es estático

# Cómputos ≡ evaluación de expresiones (III)

```
> 3+3 == 2*3
True
> True && False == True
False
> 3+3 == True
Error de
tipo
```

`==` función de igualdad, polimórfica

Los dos lados de `==` han de tener el mismo tipo

```
> 3+3 /= 4
True
> 3+3 /= True
Error de
tipo
```

`/=` función de desigualdad, polimórfica

Los dos lados de `/=` han de tener el mismo tipo

# Funciones predefinidas sobre enteros, reales, booleanos,...

Muchas de ellas están sobrecargadas para distintos tipos
$\rightsquigarrow$ Más explicaciones al ver *clases de tipos*

- `+, -, *, /, div, mod, ^, ^^, **,...`
- `even, odd, lcm, gcd`
- `abs, signum, negate, min, max`
- `pi, exp, sqrt, log, **, logBase, sin, tan, cos, asin, atan, acos, ...`
- `truncate, round, ceiling, floor, fromInteger, toInteger, fromIntegral`

Probadlas todas!

# Evaluación de expresiones: tuplas

Tuplas ≡ agrupación de un número fijo de valores de cualquier tipo
$\rightsquigarrow$ tipo de datos polimórfico

```
> (1+2,True && False)
(3,False)
> (1+2,(0,1),0,succ 'a')
(3,(0,1),0,'b')
> ()
()
> fst (3,5)
3
> snd(5,True)
True
> (3,5) == (5,3)
False
> (3,True) == (True,3)
Error de tipo
> (3,5) == (3,5,5)
Error de
tipo
```

Las componentes pueden ser de distinto tipo

Hay tuplas de cualquier tamaño

Incluso tupla vacía

`fst` tiene un argumento, no dos

`snd` tiene un argumento, no dos

El orden influye en el valor de una tupla

El orden influye en el tipo de una tupla

Tuplas de distinto tamaño tienen tipos distintos

# Evaluación de expresiones: listas (I)

## Listas

- Secuencias de longitud no prefijada !incluso infinita!
- Los elementos de la lista pueden ser de cualquier tipo
  Listas $\rightsquigarrow$ tipo de datos polimórfico
- Pero todos los elementos deben ser del mismo tipo

```
> [1+2,3,5-1,7]
```
Lista de enteros
```
[3,3,4,7]
> [True,False || True]
```
Lista de booleanos
```
[True,True]
> []
```
Lista vacía (de cualquier tipo)
```
[]
> [[1,2],[],[5]]
```
Lista de listas de enteros
```
[[1,2],[],[5]]
> [True, 1, True]
Error de tipo
```
Listas polimórficas pero homogéneas

# Evaluación de expresiones: listas (II)

```
> [1,2,3] == [3,4]
False
```
Listas distintas, pero del mismo tipo

```
> [1,2] == ['a','b']
Error de tipo
```
Listas de distinto tipo

```
> [1,1] == [1]
False
```
Las repeticiones cuentan

```
> [0,1] == [1,0]
False
```
El orden cuenta

# Evaluación de expresiones: listas (III)

```
> head [1,3,5]
```
cabeza de la lista

```
1
> tail [1,2,3,4]
```
resto de la lista

```
[2,3,4]
> tail [1]
[]
```
Lista vacía de enteros

```
> tail [True]
[]
```
Lista vacía de booleanos

`[]` es una constante polimórfica

```
> tail [1] == tail [True]
Error de tipo
```
Las dos apariciones de `[]` tienen tipos distintos

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
> head [3 `div` 0,7]
```

`head` y `tail` definidas solo para listas no vacías

Aplicarlas a `[]` no es error sintáctico ni de tipo

`head` y `tail` son funciones *parciales*

Inciso: `3 `div` 0` $\equiv$ `div 3 0`

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
```

`head` y `tail` definidas solo para listas no vacías

Aplicarlas a `[]` no es error sintáctico ni de tipo

`head` y `tail` son funciones *parciales*

```
> head [3 `div` 0,7]
Exception:  divide by zero
```

Inciso: `3 `div` 0` $\equiv$ `div 3 0`

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
```

`head` y `tail` definidas solo para listas no vacías

Aplicarlas a `[]` no es error sintáctico ni de tipo

`head` y `tail` son funciones *parciales*

```
> head [3 `div` 0,7]
Exception:  divide by zero
> head [1,5,3 `div` 0,7]
```

Inciso: `3 `div` 0` $\equiv$ `div 3 0`

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
```

`head` y `tail` definidas solo para listas no vacías

Aplicarlas a `[]` no es error sintáctico ni de tipo

`head` y `tail` son funciones *parciales*

```
> head [3 `div` 0,7]
Exception:  divide by zero
> head [1,5,3 `div` 0,7]
1
> tail [3 `div` 0,1,5,7]
```

Inciso: `3 `div` 0` ≡ `div 3 0`

`Haskell realiza evaluación perezosa`

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
```

`head` y `tail` definidas solo para listas no vacías

Aplicarlas a `[]` no es error sintáctico ni de tipo

`head` y `tail` son funciones *parciales*

```
> head [3 `div` 0,7]
Exception:  divide by zero
> head [1,5,3 `div` 0,7]
1
> tail [3 `div` 0,1,5,7]
[1,5,7]
> tail [1,5,3 `div` 0,7]
```

Inciso: `3 `div` 0` $\equiv$ `div 3 0`

Haskell realiza evaluación perezosa

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
```

`head` y `tail` definidas solo para listas no vacías

Aplicarlas a `[]` no es error sintáctico ni de tipo

`head` y `tail` son funciones *parciales*

```
> head [3 `div` 0,7]
Exception: divide by zero
> head [1,5,3 `div` 0,7]
1
> tail [3 `div` 0,1,5,7]
[1,5,7]
> tail [1,5,3 `div` 0,7]
[5,Exception
```

Inciso: `3 `div` 0` ≡ `div 3 0`

Haskell realiza evaluación perezosa

Construcción monótona del resultado

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
```

`head` y `tail` definidas solo para listas no vacías

Aplicarlas a `[]` no es error sintáctico ni de tipo

`head` y `tail` son funciones *parciales*

```
> head [3 `div` 0,7]
Exception: divide by zero
> head [1,5,3 `div` 0,7]
1
> tail [3 `div` 0,1,5,7]
[1,5,7]
> tail [1,5,3 `div` 0,7]
[5,Exception
> tail [5,3 `div` 0,7]
[Exception
```

Inciso: `3 `div` 0` ≡ `div 3 0`

Haskell realiza evaluación perezosa

Construcción monótona del resultado

Construcción monótona del resultado

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
> head [3 `div` 0,7]
Exception: divide by zero
> head [1,5,3 `div` 0,7]
1
> tail [3 `div` 0,1,5,7]
[1,5,7]
> tail [1,5,3 `div` 0,7]
[5,Exception
> tail [5,3 `div` 0,7]
[Exception
> tail [1,tail [],4]
```

head y tail definidas solo para listas no vacías

Aplicarlas a [] no es error sintáctico ni de tipo

head y tail son funciones *parciales*

Inciso: 3 `div` 0 ≡ div 3 0

Haskell realiza evaluación perezosa

Construcción monótona del resultado

Construcción monótona del resultado

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
```

`head` y `tail` definidas solo para listas no vacías

Aplicarlas a `[]` no es error sintáctico ni de tipo

`head` y `tail` son funciones *parciales*

```
> head [3 `div` 0,7]
Exception: divide by zero
> head [1,5,3 `div` 0,7]
1
> tail [3 `div` 0,1,5,7]
[1,5,7]
> tail [1,5,3 `div` 0,7]
[5,Exception
> tail [5,3 `div` 0,7]
[Exception
> tail [1,tail [],4]
Error de tipo
```

Inciso: `3 `div` 0` ≡ `div 3 0`

Haskell realiza evaluación perezosa

Construcción monótona del resultado

Construcción monótona del resultado

`tail []` tiene el tipo de una lista

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
> head [3 `div` 0,7]
Exception: divide by zero
> head [1,5,3 `div` 0,7]
1
> tail [3 `div` 0,1,5,7]
[1,5,7]
> tail [1,5,3 `div` 0,7]
[5,Exception
> tail [5,3 `div` 0,7]
[Exception
> tail [1,tail [],4]
Error de tipo
> head (tail (tail [1,head [],4]))
```

head y tail definidas solo para listas no vacías

Aplicarlas a [] no es error sintáctico ni de tipo

head y tail son funciones *parciales*

Inciso: 3 `div` 0 ≡ div 3 0

Haskell realiza evaluación perezosa

Construcción monótona del resultado

Construcción monótona del resultado

tail [] tiene el tipo de una lista

# Evaluación de expresiones: listas (IV)

```
> head []
Exception
> tail []
Exception
```

`head` y `tail` definidas solo para listas no vacías

Aplicarlas a `[]` no es error sintáctico ni de tipo

`head` y `tail` son funciones *parciales*

```
> head [3 `div` 0,7]
Exception: divide by zero
> head [1,5,3 `div` 0,7]
1
> tail [3 `div` 0,1,5,7]
[1,5,7]
> tail [1,5,3 `div` 0,7]
[5,Exception
> tail [5,3 `div` 0,7]
[Exception
> tail [1,tail [],4]
Error de tipo
> head (tail (tail [1,head [],4]))
4
```

Inciso: `3 `div` 0` ≡ `div 3 0`

Haskell realiza evaluación perezosa

Construcción monótona del resultado

Construcción monótona del resultado

`tail []` tiene el tipo de una lista

evaluación perezosa