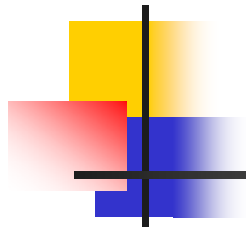


Programación Funcional

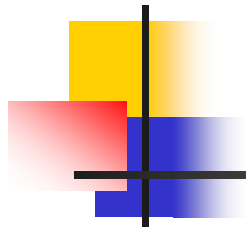
Curso 2017/18

Marisa Navarro



Bibliografía Básica

- Bird, R.
“Introducción a la Programación Funcional con Haskell”
(2ª.ed), Prentice Hall. 1998 (traducción en español: 2000)
- Thompson, S.
“Haskell.The Craft of Functional Programming” (2ª.ed),
Addison-Wesley. 1999.
- Home page de Haskell: <http://www.haskell.org/>



Temario

Tema 1. Introducción.

Tema 2. T.D. simples. Definición de funciones. Currificación.

Tema 3. Constructores de tipos. Ajuste de patrones. Polimorfía.

Tema 4. Funciones sobre listas. Orden superior.

Tema 5. Operadores sobrecargados. Clases de tipos.

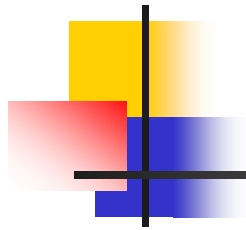
Tema 6. Tipos algebraicos. Inducción estructural.

Tema 7. Tipos abstractos de datos. Módulos.

Tema 8. Evaluación perezosa. Listas infinitas.

Tema 9. Eficiencia. Estructuras cíclicas.

Tema 10. Programando con acciones: I/O.



Tema 1. Introducción

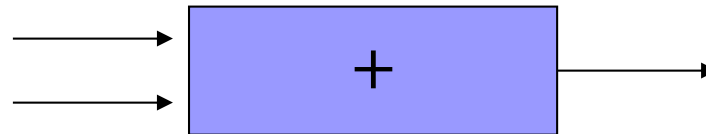
Programación Funcional:

- Consiste en construir definiciones de funciones y usar el ordenador para evaluar expresiones.
- Papel del programador:
construir las definiciones de funciones que resuelvan un problema dado.

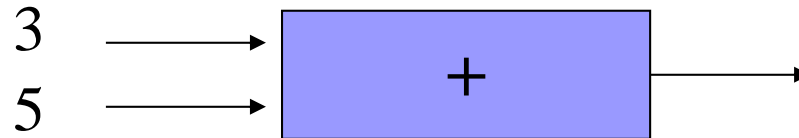


Funciones como cajas negras (1)

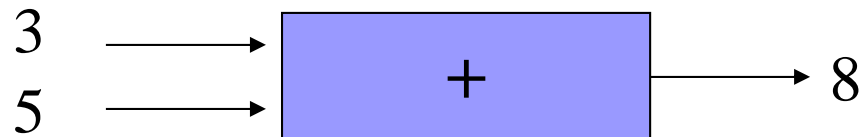
- ¿Qué es una función? Una “caja negra”



- ¿Aplicación de una función? Meter argumentos/entradas

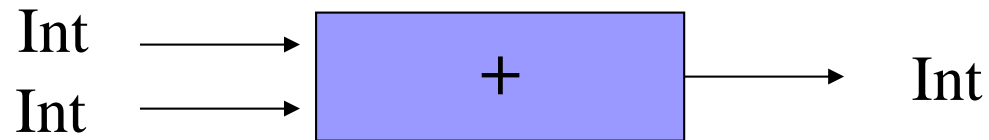


- ¿Evaluación de una expresión? Obtener el resultado/salida



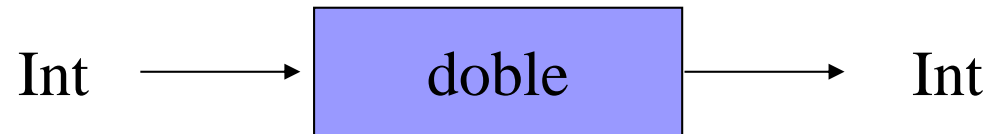
Funciones como cajas negras (2)

- ¿Tipo de una función?



- ¿Definición de una función? Interior de la “caja negra”

Ejemplo:



Posibles definiciones:

(a) $\text{doble } x = x + x$

(b) $\text{doble } x = x * 2$

Evaluación correspondiente:

$\text{doble } 7 \rightarrow 7 + 7 \rightarrow 14$

$\text{doble } 7 \rightarrow 7 * 2 \rightarrow 14$

¡Una función puede tener distintas definiciones!

Funciones como cajas negras (3)

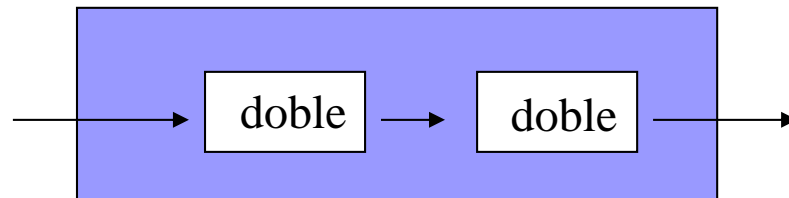
Ejemplo: función “cuádruple”



“cuádruple x = el número x cuádruplicado”

Una posible definición: **cuádruple x = doble (doble x)**

Interior de la caja “cuádruple”:

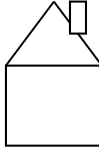


Evaluación:

cuádruple 7 \Rightarrow doble (doble 7) \Rightarrow doble 14 \Rightarrow 28

Ejemplo de introducción (1)

Suponemos un tipo “Dibujo” y una constante de este tipo:

casa = 

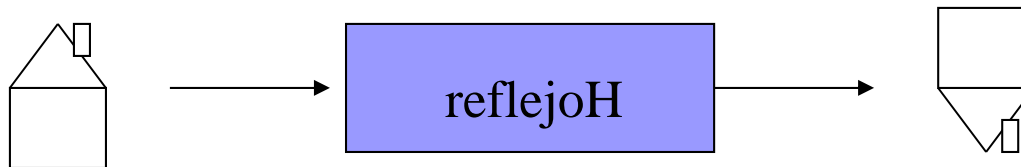
casa :: Dibujo

Funciones sobre el tipo Dibujo:

reflejoV :: Dibujo → Dibujo

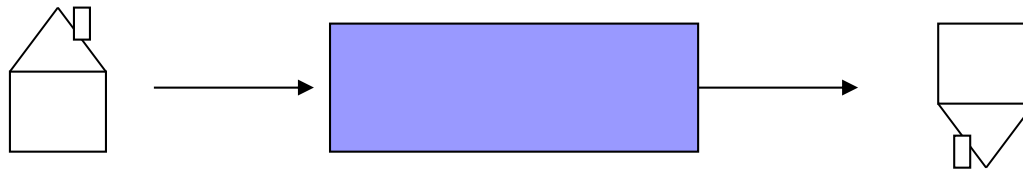


reflejoH :: Dibujo → Dibujo



Ejemplo de introducción (2)

Nueva función: $\text{rotar} :: \text{Dibujo} \rightarrow \text{Dibujo}$

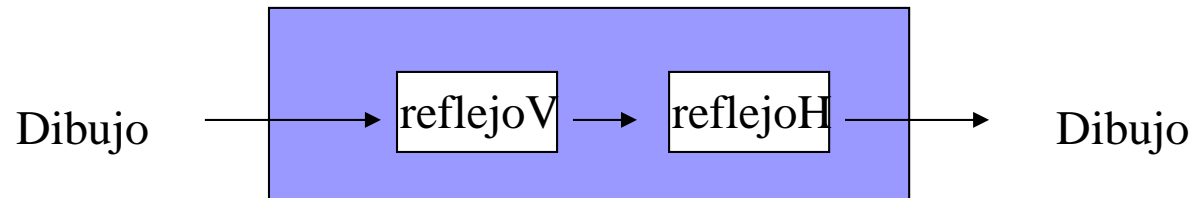


¿Cómo definir “rotar” en términos de las otras funciones?

$\text{rotar dib} = \text{reflejoH} (\text{reflejoV dib}) = (\text{reflejoH} \bullet \text{reflejoV}) \text{ dib}$

También se puede escribir directamente como:

$\text{rotar} = \text{reflejoH} \bullet \text{reflejoV}$





Ejemplo de introducción (3)

Implementación del tipo Dibujo:

```
type Dibujo = [Linea]           -- lista de líneas
type Linea = [Char]             -- lista de caracteres
```

Implementación de las funciones básicas del tipo Dibujo:

(usar “reverse” para invertir listas y “map” que aplica una función a todos los elementos de una lista)

```
reflejoH :: Dibujo → Dibujo
```

```
reflejoH d =
```

```
reflejoV :: Dibujo → Dibujo
```

```
rerflejoV d =
```



Conclusiones del ejemplo

- ❑ Uso de “reverse” para invertir un dibujo (lista de líneas) y para invertir una línea (lista de caracteres)
 - Polimorfía $\text{reverse} :: [\alpha] \rightarrow [\alpha]$
- ❑ Uso de “reverse” como argumento de la func. “map”
 - Función de Orden Superior
$$\text{map} :: (\alpha \rightarrow \beta) \rightarrow [\alpha] \rightarrow [\beta]$$
 - Funciones como ciudadanos de primera clase
- ❑ Propiedad que cumple nuestro programa:
$$\text{“reflejoH} \bullet \text{reflejoV} = \text{reflejoV} \bullet \text{reflejoH} \text{”}$$
 - Razonamiento formal
 - Aplicación: Transformación de programas



Características y beneficios de la P.F.

- Tipado fuerte => Errores detectados en compilación
 - Polimorfismo => Reutilización de código
 - Evaluación perezosa => Modularidad, reusabilidad
 - Func. orden superior => Abstracciones poderosas
 - Gestión de memoria interna
-
- Incremento en la productividad del programador
 - Menor salto semántico entre programador y lenguaje
 - Programas más cortos, claros y fáciles de entender
 - Mejor mantenimiento y mayor fiabilidad



Lenguaje Haskell

- Lenguaje de P.F. *más moderno* y “*de facto*” estándar
- Su nombre se debe a **Haskell Brooks Curry**
- Basado en el λ -cálculo
- Con las características usuales de la P.F. + otras: *clases de tipos (sobrecarga de operaciones), sistema de módulos, I/O monádica.*
- Usado en universidades: primer lenguaje de programación
- Usado en la industria: prototipado, diseño y verificación de hardware.
- The “Haskell home page” es: <http://www.haskell.org>

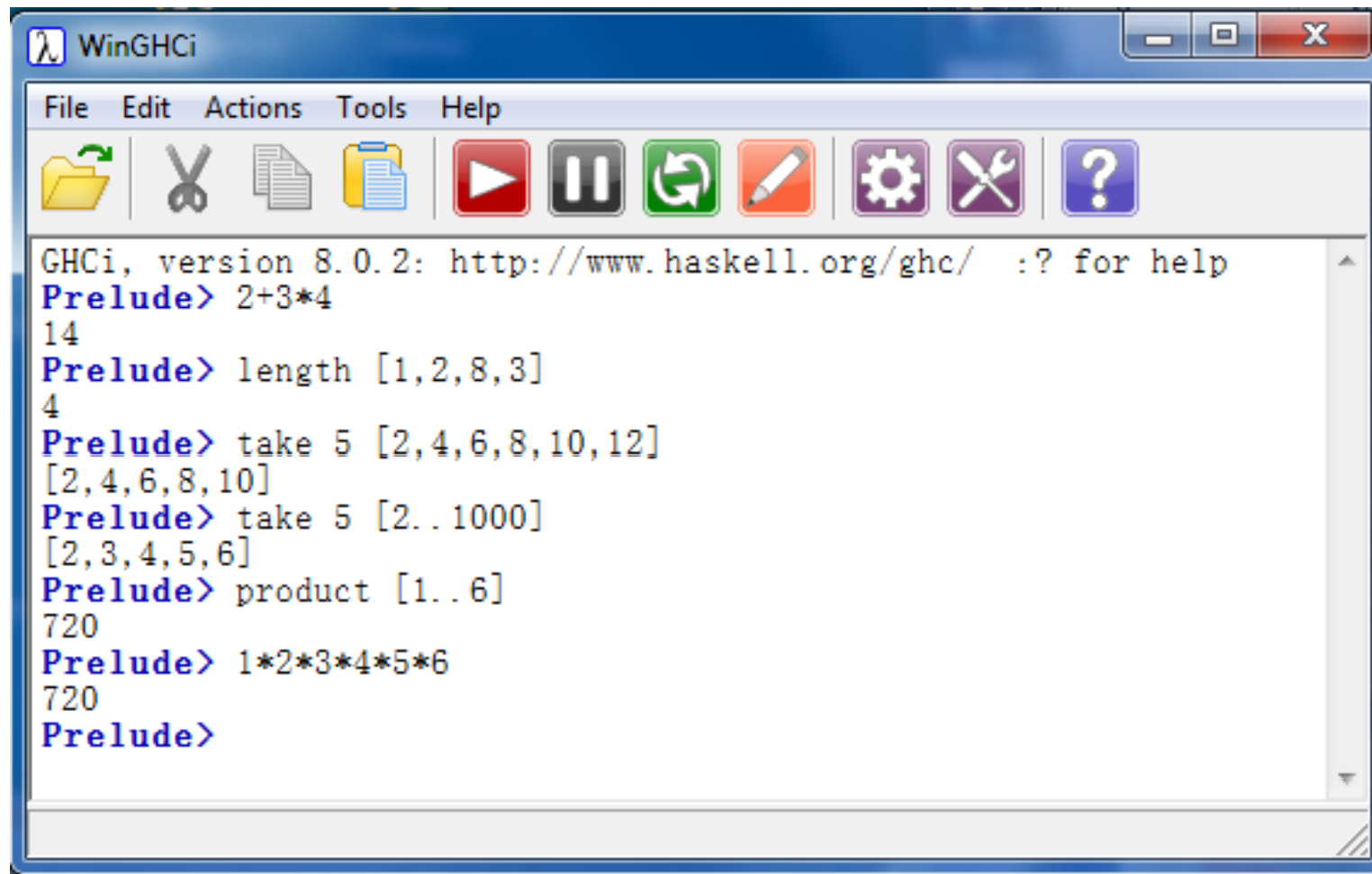


Sistema WinGHCi

- **GHCi** = entorno interactivo de Glasgow Haskell Compiler
- **WinGHCi** = implementación de Haskell que usaremos en el laboratorio.
- Una sesión en el entorno interactivo (Win)GHCi es similar a una calculadora poderosa:
 - Se *carga un **script (guión)*** o fichero con definiciones de funciones (+ declaraciones + comentarios)
 - Se le pide que *evalúe expresiones* (que usan dichas funciones)
- Tiene además funciones y operadores predefinidos (en el script **Prelude.hs** que se carga automáticamente).

Ejemplo de sesión en WinGHCi (1)

- Sólo con las funciones predefinidas:



The screenshot shows the WinGHCi application window. The title bar reads 'WinGHCi'. The menu bar includes 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for file operations (folder, copy, paste, save), execution (play, pause, refresh), editing (eraser), settings (gear), and help (question mark). The main text area displays the following session:

```
GHCi, version 8.0.2: http://www.haskell.org/ghc/  :? for help
Prelude> 2+3*4
14
Prelude> length [1,2,8,3]
4
Prelude> take 5 [2,4,6,8,10,12]
[2,4,6,8,10]
Prelude> take 5 [2..1000]
[2,3,4,5,6]
Prelude> product [1..6]
720
Prelude> 1*2*3*4*5*6
720
Prelude>
```

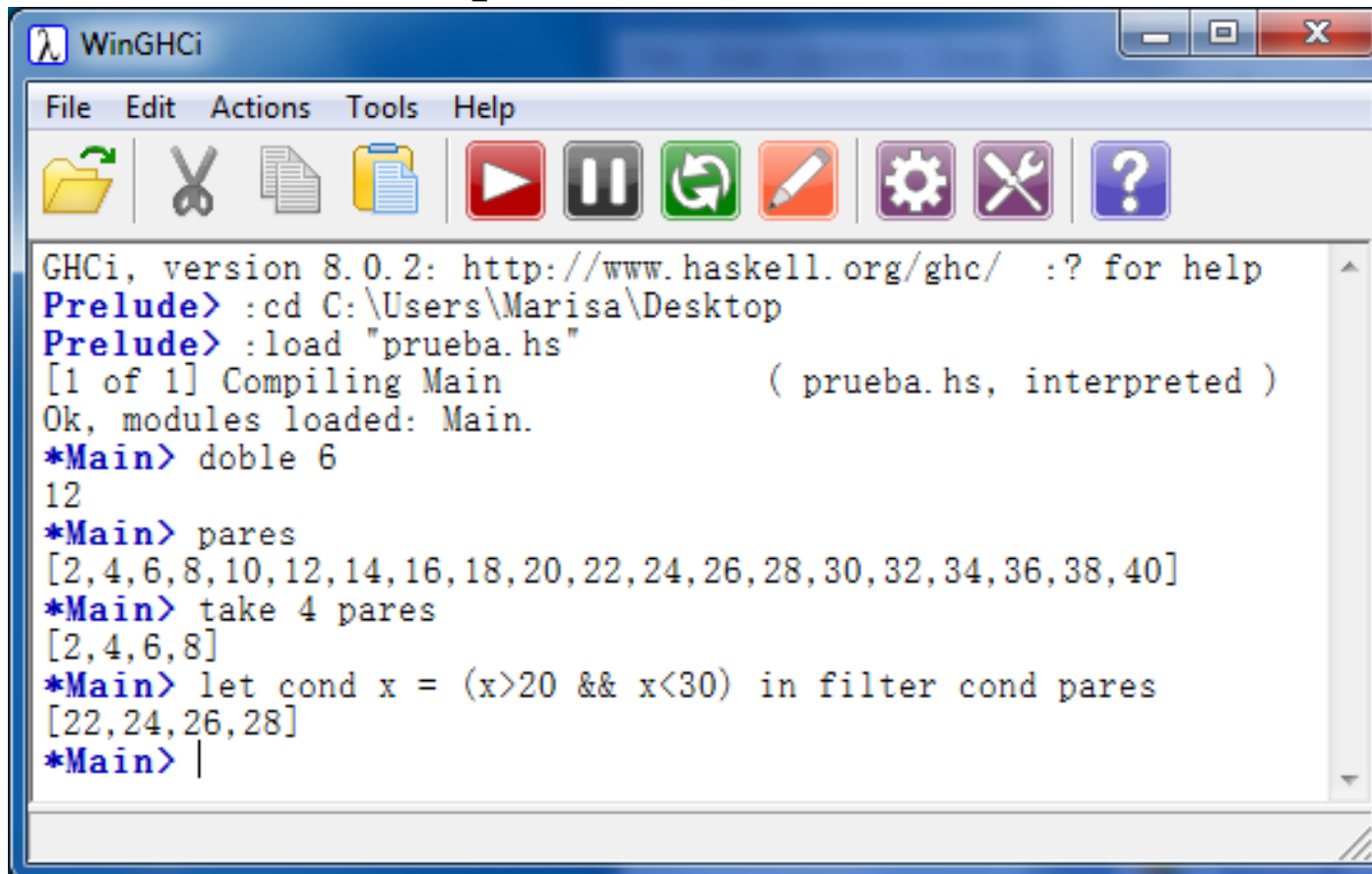
Ejemplo de sesión en WinGHCi (2)

➤ Si el script *prueba.hs* es

```
dobles x = x + x
```

```
pares = [dobles x | x <- [1..20]]
```

podemos evaluar expresiones como:



```
WinGHCi
File Edit Actions Tools Help
[Icons: Folder, Scissors, Document, Clipboard, Play, Pause, Refresh, Pencil, Gear, Wrench, Question Mark]
GHCi, version 8.0.2: http://www.haskell.org/ghc/  :? for help
Prelude> :cd C:\Users\Marisa\Desktop
Prelude> :load "prueba.hs"
[1 of 1] Compiling Main                ( prueba.hs, interpreted )
Ok, modules loaded: Main.
*Main> dobles 6
12
*Main> pares
[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40]
*Main> take 4 pares
[2,4,6,8]
*Main> let cond x = (x>20 && x<30) in filter cond pares
[22,24,26,28]
*Main> |
```