Programación Funcional

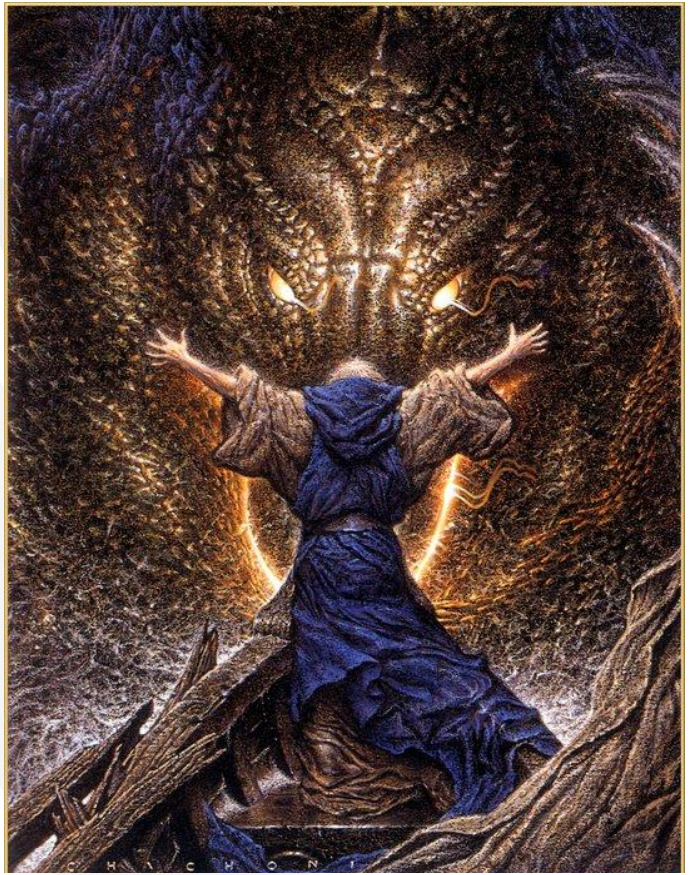
Clases teóricas

por Pablo E. “Fidel” Martínez López

1. Presentación de la materia
   Modelo funcional - Alto orden

“Cuando sepas reconocer la cuatrifolia en todos sus estados, raíz, hoja y flor, por la vista y el olfato, y la semilla, podrás aprender el verdadero nombre de la planta, ya que entonces conocerás su esencia, que es más que su utilidad.”

Un mago de Terramar
Úrsula K. Le Guin



Conceptos preliminares

Objetivos de la materia

- ¿Cuál es la esencia de la programación?
  - Conocer características del Paradigma Funcional nos acerca a esa esencia
  - No buscamos una aproximación formal
  - Buscamos dar un panorama de **conceptos relevantes**

Definición de programa

- ¿Cómo definimos programa?

Definición de programa

- ¿Cómo definimos programa?
  - Definición usual:
    secuencia de instrucciones para lograr un objetivo

Definición de programa

- ¿Cómo definimos programa?
  - Definición usual:
    secuencia de instrucciones para lograr un objetivo
  - Problema:
    ¡en Programación Funcional NO HAY instrucciones!

Definición de programa

- ¿Cómo definimos programa?
  - Definición usual:
    secuencia de instrucciones para lograr un objetivo
  - Problema:
    ¡en Programación Funcional NO HAY instrucciones!
  - Precisamos una definición de programa más amplia

Definición de programa

- **Programa:**
  - **descripción** de una solución a un problema **ejecutable** por algún mecanismo automatizable

Definición de programa

- **Programa:**
  - **descripción** de una solución a un problema **ejecutable** por algún mecanismo automatizable
  - Solo para *problemas computacionales*

Definición de programa

- **Programa:**
  - **descripción** de una solución a un problema **ejecutable** por algún mecanismo automatizable
  - Solo para *problemas computacionales*
  - Esta definición muestra la naturaleza *dual* de los programas

Problemas computacionales

- ¿Qué tipo de problemas computacionales?
  - *transformación de información*
    - datos de entrada en datos de salida
  - *interacción con el medio*
    - interfaces, periféricos, otros programas, etc.
  - En la primera parte del curso solo nos ocuparemos del 1ro de los tipos: transformación de información

Naturaleza de los programas

- **Programa:**
  - **descripción ejecutable**
    - Los programas son ejecutados por **máquinas**
      - **Ejecutable** hace mención a este hecho

Naturaleza de los programas

- **Programa:**
  - **descripción ejecutable**
    - Los programas son ejecutados por **máquinas**
      - **Ejecutable** hace mención a este hecho
    - Pero también son leídos y entendidos por **personas**
      - **Descripción** hace mención a este otro

Naturaleza de los programas

- **descripción ejecutable**

- El programa en tanto *descripción*
  - Es un ***TEXTO***
  - Importa *qué* describe y no cómo lo hace
  - Decimos que es una visión de *alto nivel*
  - También decimos que es una visión ***denotacional***
  - Diferentes programas pueden describir lo mismo
    - **Equivalencia** de programas

Naturaleza de los programas

- ❏ **descripción ejecutable**

- ❏ El programa en tanto su naturaleza *ejecutable*
  - ❏ Importa *cómo* ejecuta y los recursos que usa
  - ❏ Decimos que es una visión de *bajo nivel*
  - ❏ También decimos que es una visión ***operacional***
  - ❏ ¡Dos programas equivalentes denotacionalmente pueden *ser diferentes operacionalmente*!
    - ❏ Debemos definir el modelo de ejecución

Equivalencia de programas

- ¿Es fácil saber si dos programas son equivalentes?
  - ¿Son equivalentes `f(3)+f(3)` y `2*f(3)`?
  - ¿Siempre?
  - ¿Deberían serlo?
  - ¿Por qué?

Equivalencia de programas

- ¿Qué imprime este programa en Javascript?

```
// Test.js     (by Martín Goffan, 2018)
let x = 0;
function f(y) {   x = x + 1;
                  return x + y; }
console.log(f(3)+f(3));
```

  - ¿Y con `2*f(3)` en lugar de `f(3)+f(3)`?

Equivalencia de programas

- ¿Qué imprime este programa en Java?

```
public class Example { // (by Juan Li Puma 2018)
    static int x = 0;
    static int f(int y) { return ++x + y;  }
    public static void main(String[] args)
       {  System.out.printf(f(3) + f(3));  }
}
```

  - ¿Y con `2*f(3)` en lugar de `f(3)+f(3)`?

Equivalencia de programas

- ¿Qué imprime este programa en Pascal?

```
Program test;
   var x : integer;
   function f(y:integer):integer;
      begin  x := x+1; f :=x+y;  end;
   begin  x := 0; writeln(f(3)+f(3));  end;
```

  - ¿Y con `2*f(3)` en lugar de `f(3)+f(3)`?

Equivalencia de programas

- ¿Qué precisamos para que la equivalencia de programas sea más sencilla de lograr?
  - Un **modelo de cómputo** que se concentre en el aspecto *denotacional* de los programas
  - La ejecución, el aspecto *operacional*, debe estar supeditada al significado denotacional

Modelo de cómputo funcional

- El modelo de cómputo von Neumann es útil para el aspecto operacional, pero complica el denotacional
- Un modelo de cómputo funcional estará basado en la idea de *descripción*
  - Expresiones describiendo valores

Visión denotacional de un programa

- ¿Qué es lo que queremos describir?
  - *Valores matemáticos* que representen información y transformaciones de información
- ¿Qué es un "*valor matemático*"?
  - Entidad abstracta, sin contraparte física (no tiene color, ni peso, ni forma, etc.)
  - ¡No podemos más que imaginarlos!

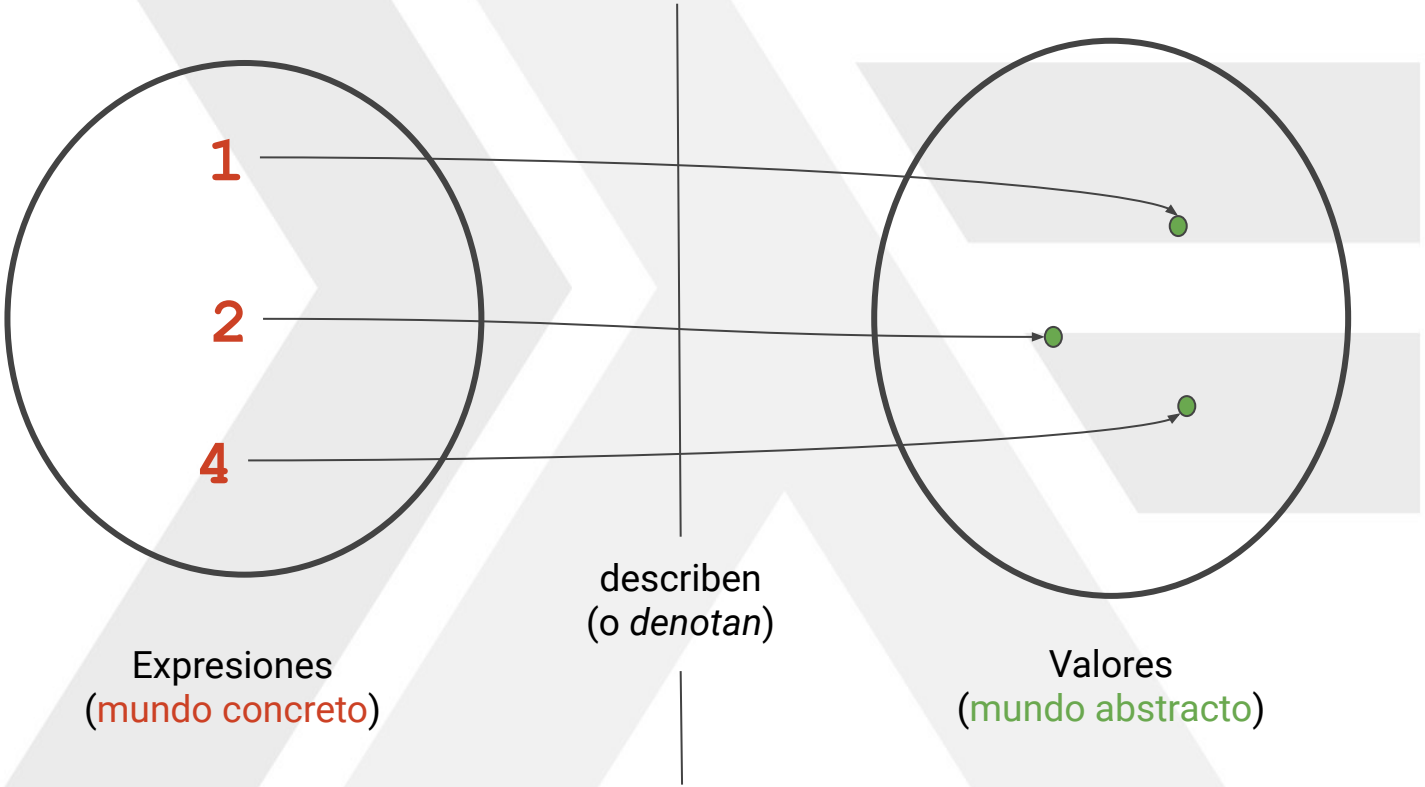
Visión denotacional de un programa

- ¿Cómo podemos describir los *valores*?
  - Mediante *expresiones*, grupos de símbolos que describen el valor que se busca significar
- ¿Cualquier grupo de símbolos es una "*expresión*"?
  - No. Hay reglas que permiten saber qué expresiones son válidas, y qué significan (qué valor describen)

Visión denotacional de un programa

- Ejemplo
  - Los números son valores
    - E.g. el número uno, el número dos, el número cuatro
  - Las expresiones más básicas que describen (denotan) números se denominan *numerales*
    - E.g. `1`, `2`, `4`

Visión denotacional de un programa



Modelo de cómputo funcional

- En ocasiones es útil distinguir un número, de la expresión que lo describe (e.g. el numeral)
- En otras ocasiones, es útil entenderlos como lo mismo
  - Decimos que 2 ***es*** el número dos
    (abusando de la notación, pues en realidad es un símbolo)
    - Podríamos escribirlo como 2 abusando de la notación
- *Identificamos* el numeral con el número
  - En el sentido de darles la misma identidad

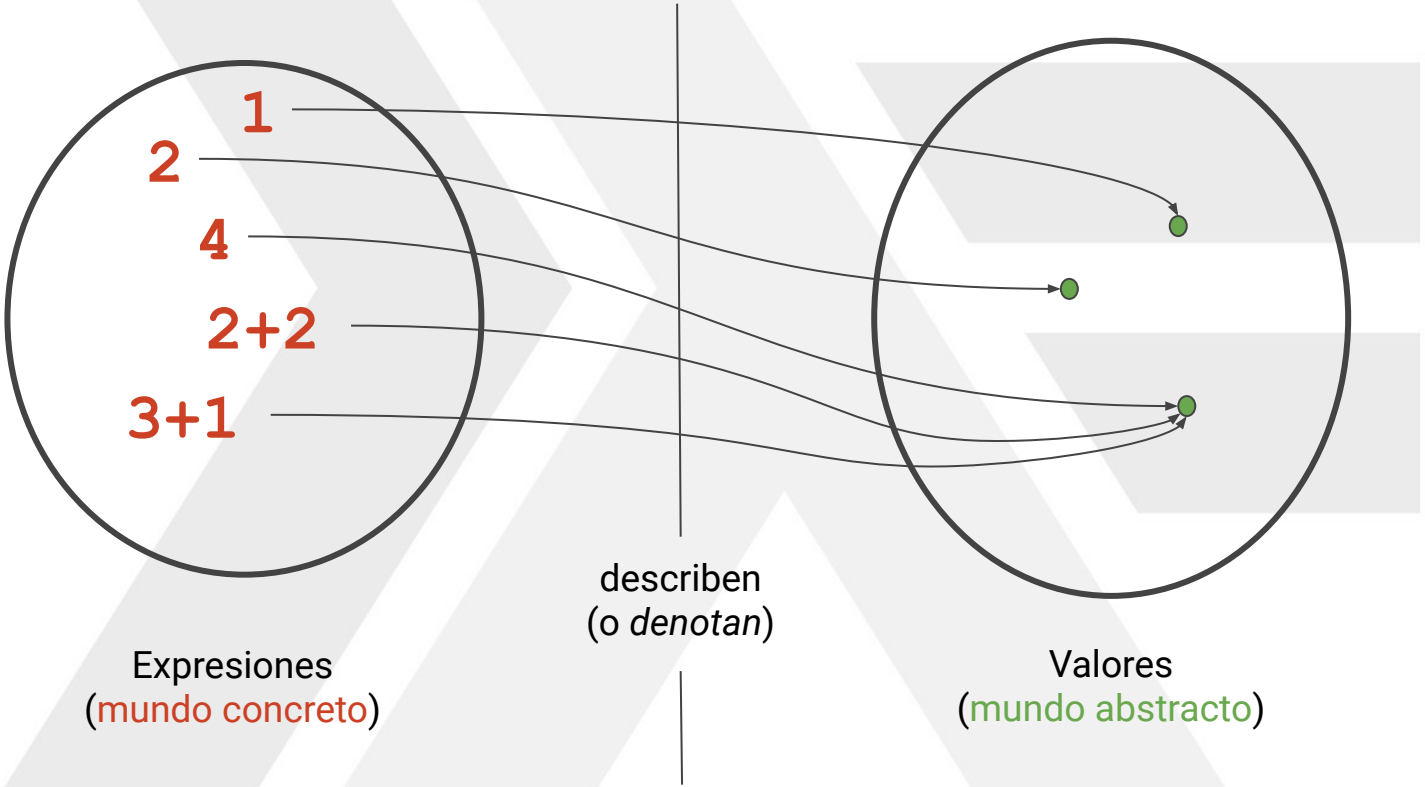
Modelo de cómputo funcional

- ¡El símbolo puede ser cualquiera!
  - E.g. los romanos usaban II, los hindúes, ২, los árabes ٢, y los habitantes de Java, 
- No podemos saber el significado de un símbolo si no lo establecemos de antemano
- Hablamos de las *convenciones* elegidas para el uso de ciertos símbolos, que dependen de cada cultura

Visión denotacional de un programa

- ¿Hay otras expresiones para denotar al cuatro?
  - ¡Sí! De hecho, hay infinitas…
  - Algunos ejemplos de expresiones que denotan el mismo valor que el símbolo `4`
    - `2+2`, `(1+1)+2`, `3+1`, etc.
  - Sin embargo, `4` es la expresión más simple
    - Decimos que es una *expresión atómica*

Visión denotacional de un programa



Visión denotacional de un programa

- Todas las expresiones que describen al mismo valor, decimos que son ***equivalentes***
- Usaremos el símbolo = para hablar de equivalencia
  - E.g. `4 = 2+2`, `3+1 = 2+2`, `3+1 = 4`,
  - Este símbolo indica que todas describen lo mismo
    - Así, puede usarse cualquiera de ellas, indistintamente

Funciones

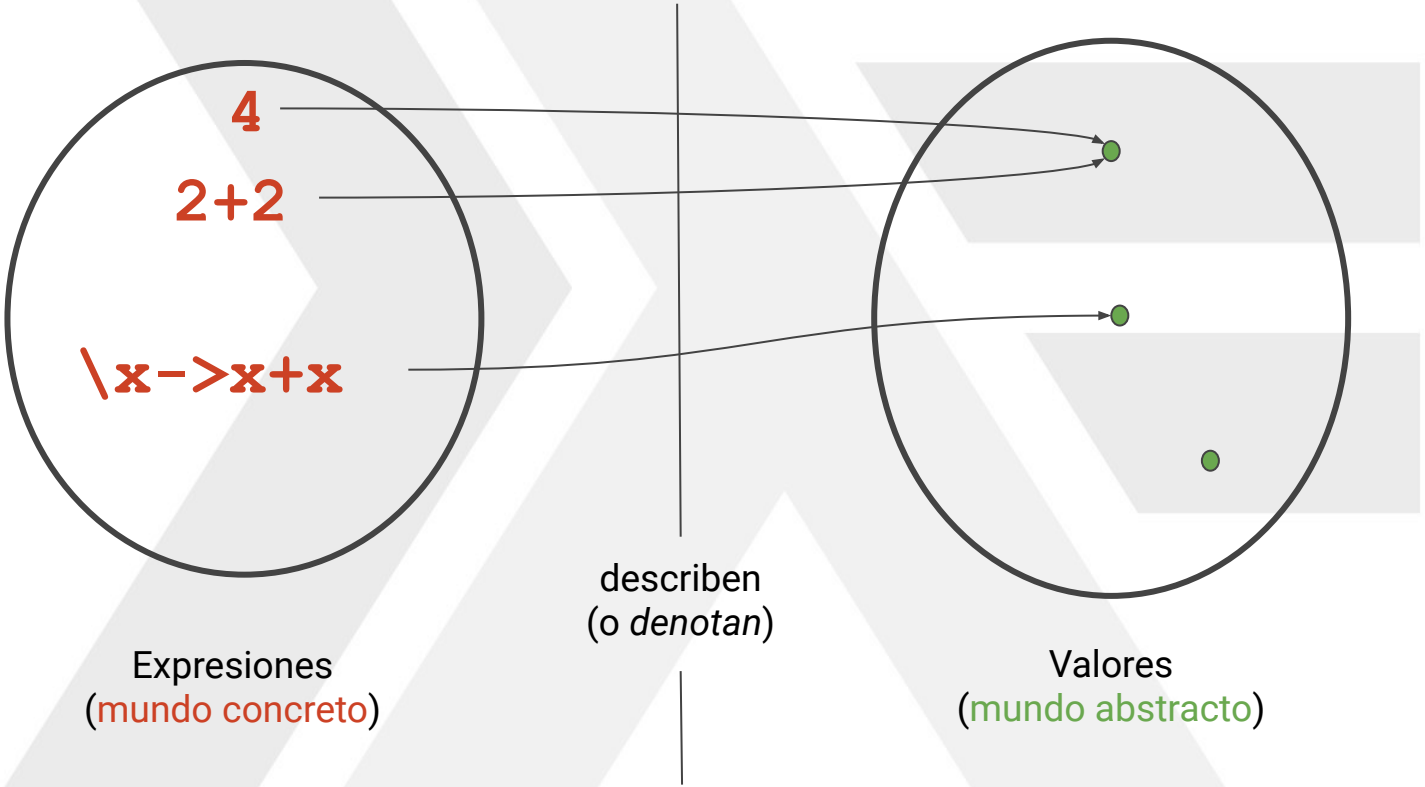
Funciones

- ¿Y para las transformaciones de información?
  - ¡Usamos funciones!
  - Las funciones transforman datos (información)
  - Datos de entrada -> datos transformados (de salida)
- ¿Cómo describimos funciones?
  - Precisamos *expresiones* que denoten funciones
  - Usaremos 3 formas diferentes para esto

Funciones

- La primera forma de expresión para funciones es específica para funciones:
  - Se denominan ***funciones anónimas***
  - Esta expresión denota directamente a una función
  - E.g. `\x -> x+x`
  - Se lee *"soy la función que transforma el dato de entrada* `x` *en el dato denotado por* `x+x`*"*

Visión denotacional de un programa



Funciones

- La segunda forma de expresión para funciones usa *nombres* que deben definirse:
  - Se requiere dar una ***definición***
  - El nombre también denota directamente a una función
    - E.g. `doble`
  - Se define mediante una ***ecuación***
    - `doble x = x+x`
  - Estas ecuaciones deben cumplir ciertas reglas

Definición mediante ecuaciones

- Reglas que deben cumplir las ecuaciones
  - Deben ir en un archivo de extensión **.hs**
  - Deben estar *orientadas*
    - Esto es, el *lado izquierdo* debe ser un nombre aún *sin significado* aplicado a nombres de argumentos, y
    - el *lado derecho* debe ser una expresión *con significado* (que puede usar los argumentos)
  - Deben seguir las reglas de sintaxis del lenguaje

Definición mediante ecuaciones

- Un archivo de definiciones se conoce como *script*
  - La extensión **.hs** es por *Haskell Script*
  - Puede contener muchas definiciones

miPrimerScript.hs

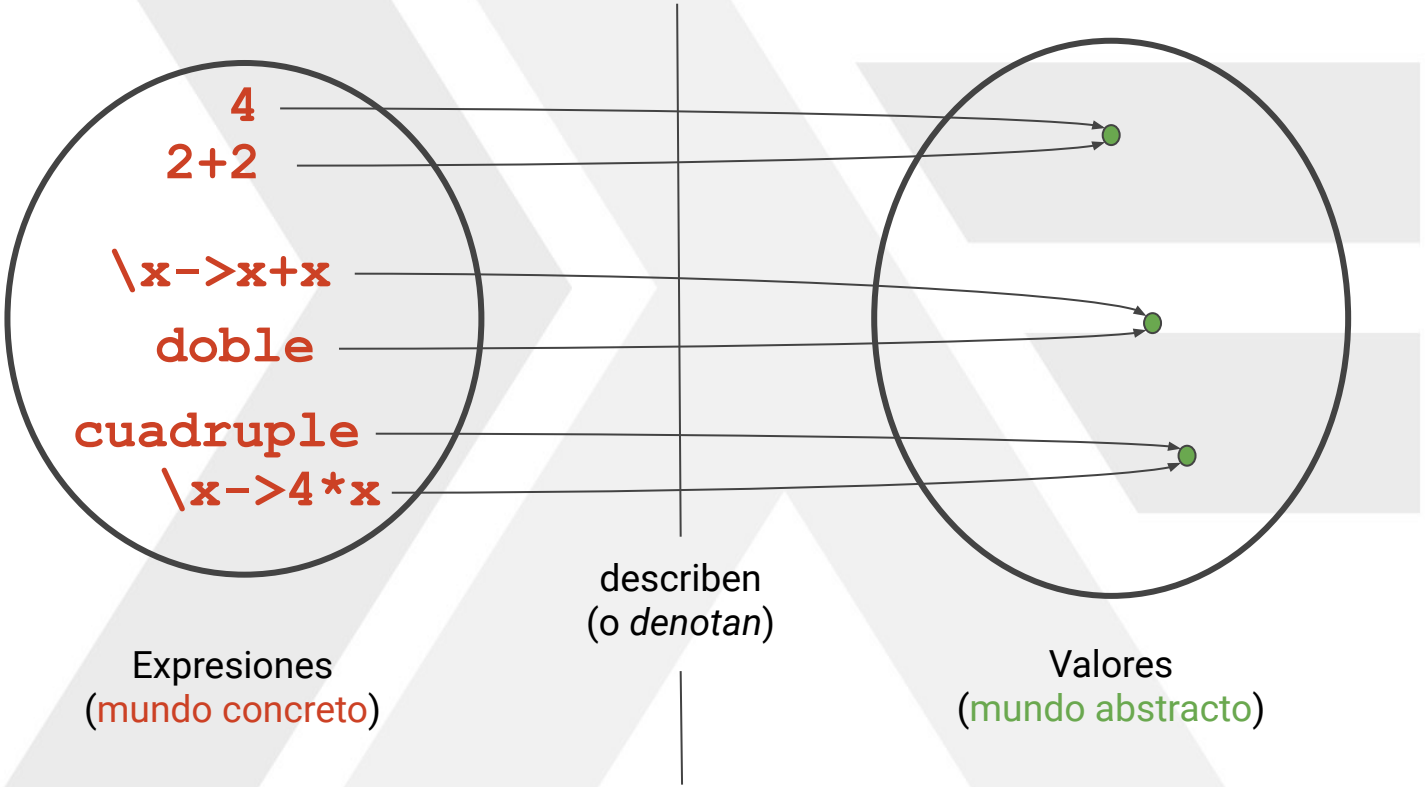
```
doble x = x+x
cuadruple x = 4*x
```

  - Los nombres a la izquierda de una ecuación denotan un valor según indica la ecuación

Definición mediante ecuaciones

- Suponiendo las definiciones anteriores…
  - `doble` denota a la función que duplica su argumento
  - `cuadruple`, a la que cuadruplica su argumento
- O sea, valen las siguientes equivalencias
  - `doble = \x->x+x`, `cuadruple = \x->4*x`
  - ¡Prestar atención a los colores!

Definición mediante ecuaciones



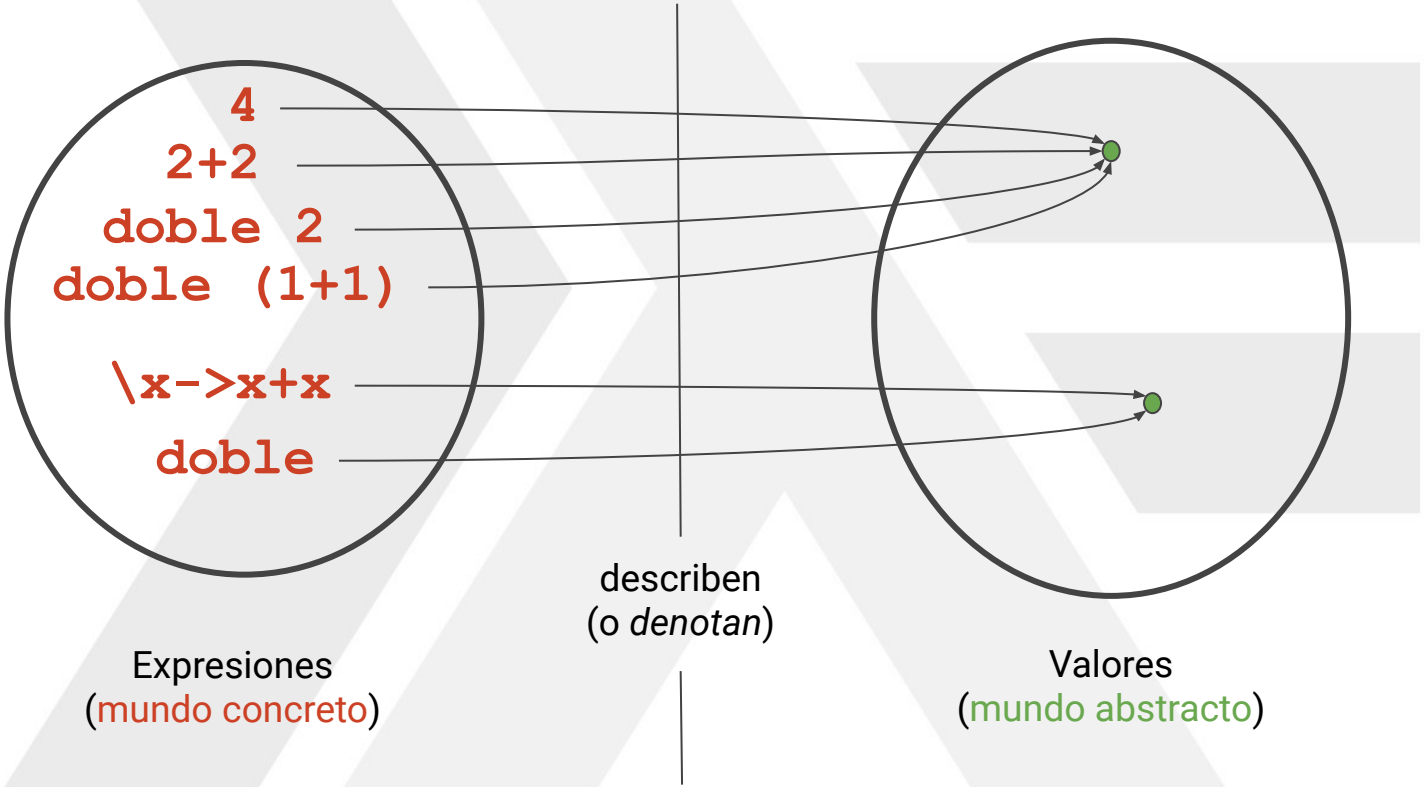
Aplicación de funciones

- ¿Cómo se usa una función?
  - Hay que suministrarle datos para transformar
  - La función se ***aplica*** a su argumento
- ¿Cómo se escribe la aplicación de una función?
  - La expresión argumento va luego de la función
    - E.g. `doble 2`, `cuadruple 3`, `(\x->x+x) 1`
  - La aplicación denota el dato transformado
    - E.g. `doble 2 = 4`, `cuadruple 3 = 12`

Aplicación de funciones

- Cualquier expresión equivalente sirve
  - E.g. `doble 2`, `doble (1+1)`
- Si a una función le damos expresiones equivalentes, el valor denotado es el mismo
  - E.g. `doble 2 = 4`, `4 = doble (1+1)`
  - Y también `doble 2 = doble (1+1)`

Aplicación de funciones



Visiones denotacional y operacional

Significado mediante equivalencia

- El significado de una expresión es el valor que denota
  - Así, el significado de `doble 2` es `4`
    - En realidad, el valor cuatro, denotado también por el numeral `4`, pero abusamos de la nomenclatura
  - ¿Cuál es el significado de las siguientes expresiones?
    - `cuadruple 2`
    - `doble (doble 2)`
    - `(\x->x+x) ((\x->x+x) 2)`

Significado mediante equivalencia

- La **equivalencia** es una forma de conocer el significado de una expresión
  - El significado de `doble 2` es `4`
    y sabemos que `doble 2 = 4`
  - Todas las expresiones equivalentes tienen el mismo significado, pues denotan al mismo valor
    - Cualquiera sirve para entender el significado de otra
    - `doble 2 = 2+2`, por definición de `doble`, con `x<-2`
    - `2+2 = 4`, por definición de suma

Significado mediante equivalencia

- **Visión denotacional**
  - Importa el significado y no la expresión
  - La equivalencia es la herramienta utilizada
  - La escribimos con el símbolo = (igual verde)

> Dos expresiones son equivalentes si, y solo si, tienen el mismo significado

  - ¡Cualquier expresión equivalente sirve para describir un significado!

Significado mediante reducción

- La *visión denotacional* nos da el aspecto matemático
- ¿Cómo recuperamos el aspecto computacional?
  - Precisamos un *mecanismo de ejecución*
  - Las ecuaciones pueden ser usadas para eso
- Mecanismo de ejecución: ***reducción***
  - En una expresión, el lado izquierdo de una expresión puede reemplazarse por el derecho
  - Esto se repite hasta que no puede hacerse más

```
doble x = x+x
cuadruple x = 4*x
```

Significado mediante reducción

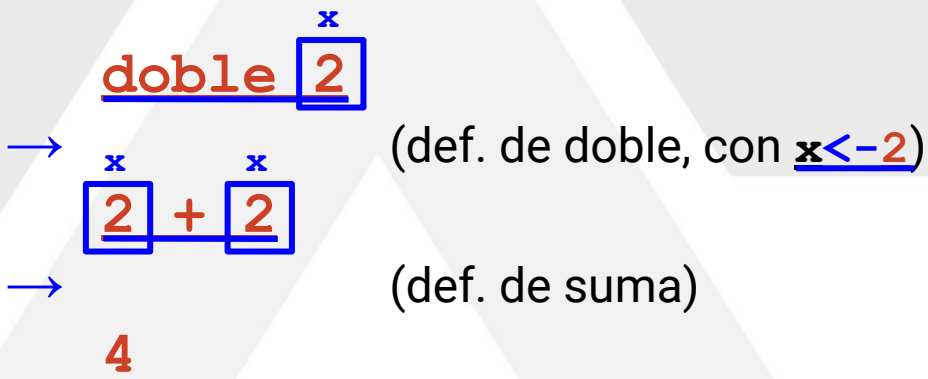
- ***Reducción***
  - Se reemplaza una subexpresión que coincide con el lado izquierdo de una ecuación, hasta no poder más
  - E.g.

```
    doble 2
→                (def. de doble, con x<-2)
    2 + 2
→                (def. de suma)
    4
```

```
doble x = x+x
cuadruple x = 4*x
```

Significado mediante reducción

- ***Reducción***
  - Se reemplaza una subexpresión que coincide con el lado izquierdo de una ecuación, hasta no poder más
  - E.g.



```
doble x = x+x
cuadruple x = 4*x
```

Significado mediante reducción

- ***Reducción***: un ejemplo con más pasos

```
   doble (doble 2)
→                          (def. de doble, con x<-2)
   doble (2 + 2)
→                          (def. de suma)
   doble 4
→                          (def. de doble, con x<-4)
   4 + 4
→                          (def. de suma)
   8
```

```
doble x = x+x
cuadruple x = 4*x
```

Significado mediante reducción

- ***Reducción***: un ejemplo con más pasos

```
    doble (doble 2)
```
(x: 2)

→ (def. de doble, con x<-2)

```
    doble (2 + 2)
```
(x: 2, 2)

→ (def. de suma)

```
    doble 4
```

→ (def. de doble, con x<-4)

```
    4 + 4
```

→ (def. de suma)

```
    8«
```

```
doble x = x+x
cuadruple x = 4*x
```

Significado mediante reducción

- ***Reducción***: un ejemplo con más pasos

```
   doble (doble 2)
→
                                  (def. de doble, con x<-2)
   doble (2 + 2)
→
                                  (def. de suma)
          x
   doble 4
→
                                  (def. de doble, con x<-4)
   x     x
   4 + 4
→
                                  (def. de suma)
   8
```

Visiones denotacional y operacional

- **Visión denotacional** **(significado)**
  - = (equivalencia de expresiones)
- **Visión operacional** **(ejecución)**
  - → (reducción)
- Vínculo entre ambas visiones
  - La reducción solo cambia expresiones equivalentes
  - Por lo tanto, el resultado de la ejecución tiene el mismo significado que el original

Visiones denotacional y operacional

- La visión operacional es menos expresiva que la visión denotacional
  - En el caso de los números, son lo mismo
  - Pero en el caso de las funciones no:
    - `cuadruple = \x -> 4*x`
    - pero no es cierto que ninguna reduzca a la otra
  - ¿Qué otros ejemplos de expresiones equivalentes que no se vinculan por reducción pueden encontrarse?

Visiones denotacional y operacional

- Funciones en visión denotacional
  - un valor matemático que relaciona cada elemento de un conjunto (de partida) con un único elemento de otro conjunto (de llegada)
- Funciones en visión operacional
  - mecanismo automatizable que dado un elemento del conjunto de partida, lo transforma en el elemento correspondiente del conjunto de llegada

Visiones denotacional y operacional

- Funciones en visión denotacional: `\x->x+x`
  - 

    soy la función que a cada dato `x` *lo relaciona con el dato* `x+x`
- Funciones en visión operacional: `\x->x+x`
  - 

    soy la función que recibe un dato `x` *y lo transforma en* `x+x`

    soy la función que toma `x` *y devuelve* `x+x`

Funciones de orden superior

Funciones de orden superior

- Las funciones también son valores
- Eso implica que pueden usarse como argumento o resultado de otras funciones
- ¿Cómo escribimos esto? ¡Con las mismas reglas!
  - E.g.

miPrimerScript.hs

```
...
twice f = g
  where g x = f (f x)
```

Funciones de orden superior

miPrimerScript.hs

```
...
twice f = g
  where g x = f (f x)
```

- Ejemplo
  - La palabra reservada **`where`** permite definiciones locales
  - La función **`g`** solo vale al usar **`twice`**
  - La función **`twice`** se usa como cualquier otra, aplicándola a valores (en este caso, funciones)
    - `twice doble`
    - `twice (\x->x+x)`
    - `twice cuadruple`
    - `twice twice`

Funciones de orden superior

miPrimerScript.hs

```
...
twice f = g
  where g x = f (f x)
```

- El resultado de twice es una función…
  - …¡entonces se puede aplicar a un valor!
  - ¿Cómo se escribe esto? ¡De la misma forma!
    - `(twice doble) 2`
  - Observar que la expresión usa una función que no es ni una expresión lambda, ni un nombre
  - Esta es la 3era forma de escribir una función

Funciones de orden superior

miPrimerScript.hs

```
...
twice f = g
  where g x = f (f x)
```

- El resultado de twice es una función…
  - …¡entonces se puede aplicar a un valor!
  - ¿Cómo se escribe esto? ¡De la misma forma!
    - `(twice doble) 2`
  - ¿Y cómo sabemos qué significa?
    - Podemos usar la equivalencia de expresiones…
    - … o (solo para números y otros valores básicos) podemos ejecutar y ver el resultado

Significado mediante reducción

```
...
twice f = g
  where g x = f (f x)
```

- ***Reducción***: uso de función de alto orden

```
   (twice doble) 2
→                                  (def. de twice, con  f<-doble)
   g 2     (donde g x = doble (doble x))
→                                  (def. de g, con x<-2)
   doble (doble 2)
→ ... →                            (varios pasos, ya realizados antes)
   8
```

¡ESTO SOLO ESTÁ PERMITIDO EN LA DIAPOSITIVA!

Significado mediante reducción

```
...
twice f = g
  where g x = f (f x)
```

- ***Reducción***: uso de función de alto orden

```
   (twice doble) 2
```
(f ← doble)

```
→  g 2     (donde g x = doble (doble x))
```
(def. de twice, con f<-doble)

```
→  doble (doble 2)
```
(def. de g, con x<-2)

```
→ ... →  8
```
(varios pasos, ya realizados antes)

¡ESTO SOLO ESTÁ PERMITIDO EN LA DIAPOSITIVA!

Significado mediante reducción

```
...
twice f = g
  where g x = f (f x)
```

- ***Reducción***: uso de función de alto orden

```
(twice doble)2
→                                  (def. de twice, con  f<-doble)
  g 2   (donde g x = doble (doble x))
→                                  (def. de g, con x<-2)
  doble (doble 2)
→ ... →                            (varios pasos, ya realizados antes)
  8
```

¡ESTO SOLO ESTÁ PERMITIDO EN LA DIAPOSITIVA!

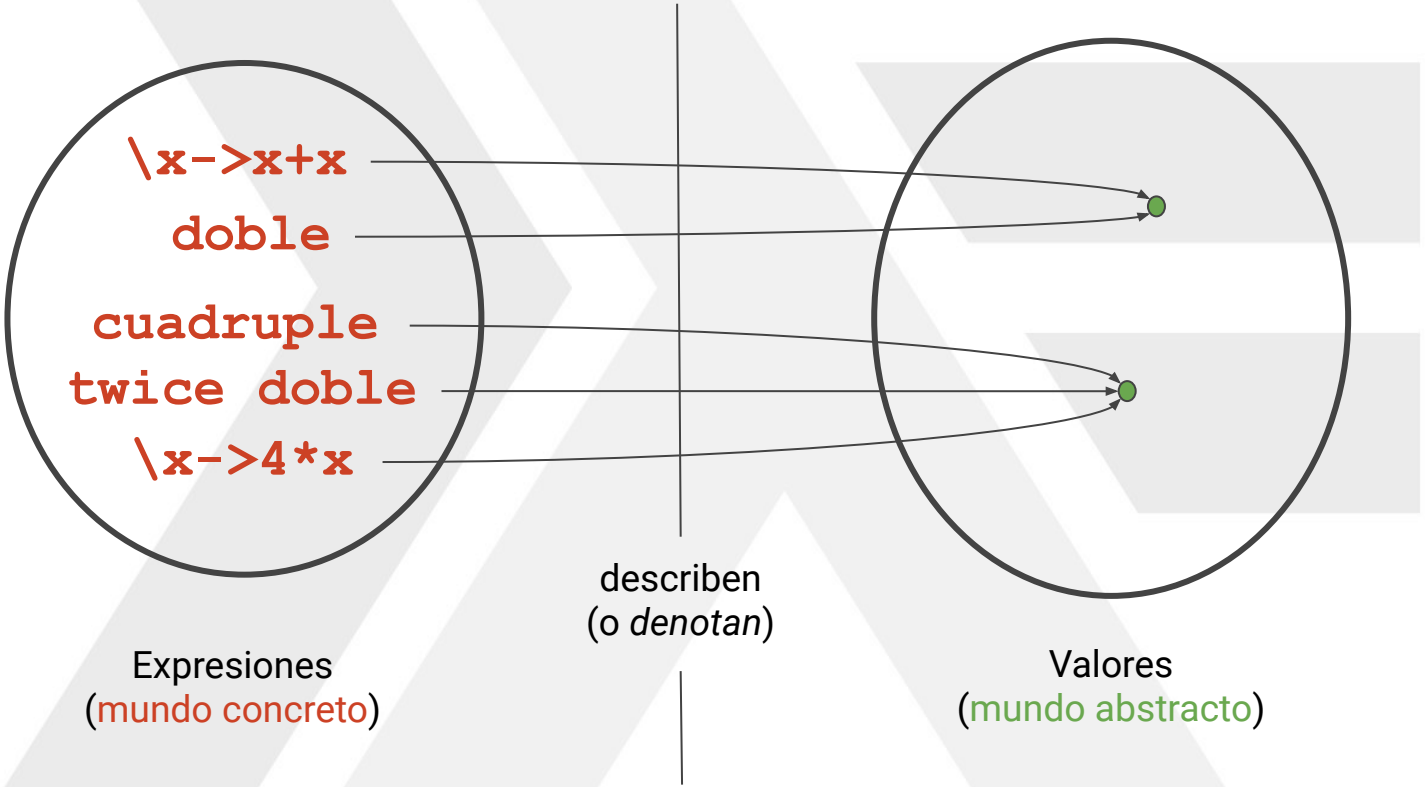
Funciones de alto orden

miPrimerScript.hs

```
...
twice f = g
  where g x = f (f x)
```

- A partir de la reducción, sabemos que
  - el significado de `(twice doble) 2` es `8`
    - `(twice doble) 2 = 8`
- Procediendo de manera similar para otros números podemos comprobar que
  - `twice doble = cuadruple`
    - pues ambas funciones transforman a todos los números de la misma manera

Aplicación de funciones



Funciones de orden superior

- La función `twice` toma una función y la transforma en otra (e.g. transforma a `doble` en `cuadruple`)
- Decimos que `twice` es una ***función de orden superior***
  - En inglés, *higher order function* (también traducido como *función de alto orden*)
  - ¡La información que transforma son funciones!

Funciones de orden superior

- ***Función de orden superior***
  - Función que toma una función como argumento y/o devuelve una función como resultado
    - Es un abuso de la nomenclatura
    - Técnicamente, la definición es un poco más precisa
  - Se pueden pesar una *estratificación* en *órdenes*

- Orden 2 - `twice`, … (transformación de funciones de orden 1)
- Orden 1 - `doble`, `cuadruple`, ... (transformación de datos básicos)
- Orden 0 - `0`,`1`,`2`,`3`, … (datos básicos)

Funciones de orden superior

- La función `twice` ***transforma*** una función
- Pero `twice` **es** una función
- ¿Podrá usarse `twice` para transformar twice?
  - Se escribiría `twice twice`
  - Es una función. ¿A qué se aplica?
  - ¡A una función!
    - E.g. `(twice twice) doble`
  - ¡¡Y esta expresión *también* es una función!!

Funciones de orden superior

- ¿Cómo se usa `(twice twice) doble`?
  - Como es una función, puede aplicarse
    - E.g. `((twice twice) doble) 2`
  - ¿Cómo saber qué significa?
    - Como siempre: equivalencia o reducción
    - Veamos como sería por reducción

Funciones de orden superior

```
...
twice f = g
  where g x = f (f x)
```

- Reducción de `((twice twice) doble) 2`

```
    ((twice twice) doble) 2
→                                   (def. de twice, con f<-twice)
    (g doble) 2          (donde g x = twice (twice x))
→                                   (def. de g, con x<-doble)
    (twice (twice doble)) 2
→                                   (def. de twice, con f<-twice doble)
    g' 2                 (donde g' x = (twice doble) ((twice doble) x))
→                                   (def. de g', con x<-2)
    (twice doble) ((twice doble) 2)
→ ...  →                            (varias reducciones, ya vistas)
    (twice doble) 8
→ ...  →                            (varias reducciones, similares a las vistas)
    32
```

¡ESTO SOLO ESTÁ PERMITIDO EN LA DIAPOSITIVA!

Funciones de orden superior

```
...
twice f = g
  where g x = f (f x)
```

- Reducción de `((twice twice) doble) 2`

```
    ((twice twice) doble) 2
→                                   (def. de twice, con f<-twice)
    (g doble) 2          (donde g x = twice (twice x))
→                                   (def. de g, con x<-doble)
    (twice (twice doble)) 2
→                                   (def. de twice, con f<-twice doble)
    g' 2                 (donde g' x = (twice doble) ((twice doble) x))
→                                   (def. de g', con x<-2)
    (twice doble) ((twice doble) 2)
→ ... →                             (varias reducciones, ya vistas)
    (twice doble) 8
→ ... →                             (varias reducciones, similares a las vistas)
    32
```

¡ESTO SOLO ESTÁ PERMITIDO EN LA DIAPOSITIVA!

Funciones de orden superior

```
...
twice f = g
  where g x = f (f x)
```

- Reducción de `((twice twice) doble) 2`

```
((twice twice) doble) 2
→                                   (def. de twice, con f<-twice)
(g doble) 2          (donde g x = twice (twice x))
→                                   (def. de g, con x<-doble)
(twice (twice doble)) 2
→                                   (def. de twice, con f<-twice doble)
g' 2                 (donde g' x = (twice doble) ((twice doble) x))
→                                   (def. de g', con x<-2)
(twice doble) ((twice doble) 2)
→ ... →                             (varias reducciones, ya vistas)
(twice doble) 8
→ ... →                             (varias reducciones, similares a las vistas)
32
```

¡ESTO SOLO ESTÁ PERMITIDO EN LA DIAPOSITIVA!

Funciones de orden superior

```
...
twice f = g
  where g x = f (f x)
```

- Reducción de `((twice twice) doble) 2`

```
((twice twice) doble) 2
→                                   (def. de twice, con f<-twice)
(g doble) 2          (donde g x = twice (twice x))
→                                   (def. de g, con x<-doble)
(twice (twice doble)) 2
→                                   (def. de twice, con f<-twice doble)
g' 2          (donde g' x = (twice doble) ((twice doble) x))
→                                   (def. de g', con x<-2)
(twice doble) ((twice doble) 2)
→ ...  →                            (varias reducciones, ya vistas)
(twice doble) 8
→ ...  →                            (varias reducciones, similares a las vistas)
32
```

¡ESTO SOLO ESTÁ PERMITIDO EN LA DIAPOSITIVA!

Funciones de orden superior

```
...
twice f = g
  where g x = f (f x)
```

- Reducción de `((twice twice) doble) 2`

```
    ((twice twice) doble) 2
→                                   (def. de twice, con f<-twice)
    (g doble) 2          (donde g x = twice (twice x))
→                                   (def. de g, con x<-doble)
    (twice (twice doble)) 2
→                                   (def. de twice, con f<-twice doble)
      x
    g' 2                 (donde g' x = (twice doble) ((twice doble) x))
→                                   (def. de g', con x<-2)
    (twice doble) ((twice doble) x 2)
→ ...  →                            (varias reducciones, ya vistas)
    (twice doble) 8
→ ...  →                            (varias reducciones, similares a las vistas)
    32
```

¡ESTO SOLO ESTÁ PERMITIDO EN LA DIAPOSITIVA!

Funciones de orden superior

- ¡La función `(twice twice)` también es una función y puede aplicarse a `twice` !
- En los estratos, `twice` es de varios órdenes

Órdenes superiores

- ⋮
- Orden 4 - `twice`, … (transformación de funciones de orden 3)
- Orden 3 - `twice`, `(twice twice) twice`, … (transformación de funciones de orden 2)
- Orden 2 - `twice`, `twice twice`, … (transformación de funciones de orden 1)
- Orden 1 - `doble`, `cuadruple`, ... (transformación de datos básicos)
- Orden 0 - `0`,`1`,`2`,`3`, … (datos básicos)

Programación funcional

- Lenguaje funcional puro
  - *Lenguaje de expresiones con funciones de orden superior, donde cada expresión tiene un único significado, y cuyo modelo de cómputo es la reducción realizada mediante el reemplazo de expresiones equivalentes*
- Programa funcional
  - *Conjunto de ecuaciones que definen el significado de una o más expresiones (la mayoría de ellas, funciones)*

Resumen

Resumen

- Programas como descripciones ejecutables
- Expresiones que describen valores
  - Equivalencia de expresiones para dar significado
- Computación mediante reemplazo de expresiones
  - Reducción de expresiones para ejecución
- Funciones para describir transformaciones de información (parámetros y aplicación de argumentos)
- Funciones como valores (funciones de orden superior)