.UBAfiuba
FACULTAD DE INGENIERÍA

75.14 / 95.48 Lenguajes Formales
Prof. Dr. Diego Corsi
Lic. Pablo Bergman

Programación Funcional

con ejemplos en FP de John Backus

Paradigmas de Programación

- **Imperativos (énfasis en la ejecución de instrucciones)**
  - Programación Procedimental (p. ej. *Pascal*)
  - Programación Orientada a Objetos (p. ej. *Smalltalk*)

- **Declarativos (énfasis en la evaluación de expresiones)**
  - **Programación Funcional** (p. ej. *Haskell*)
  - Programación Lógica (p. ej. *Prolog*)

Los lenguajes más utilizados son, en su mayoría, *multiparadigma*:
Cabe a los programadores usar el estilo de programación y las construcciones de lenguaje más adecuados para un trabajo determinado.

Características de la Programación Funcional

1. PROGRAMAS COMO FUNCIONES
2. FUNCIONES PURAS
3. DATOS INMUTABLES
4. FUNCIONES DE PRIMERA CLASE
5. FUNCIONES DE ORDEN SUPERIOR
6. COMPOSICIÓN DE FUNCIONES
7. RECURSIVIDAD

Características de la Programación Funcional

❶

PROGRAMAS COMO FUNCIONES

- **Estructura de un programa:** un programa consiste en una lista de definiciones de funciones.

- **Ejecución de un programa:** consiste en aplicar las funciones a sus argumentos (según las bases establecidas por el **Cálculo Lambda**).

Programas como funciones

Ejemplo: Cálculo del promedio de una colección de números (*prom*)

```
Def prom ≡ ÷ ∘ [/+, length]

prom : <1, 3, 5, 7>

4
```

Características de la Programación Funcional

❷

FUNCIONES PURAS

- **Determinismo:** Ante los mismos argumentos, una función pura siempre retorna el mismo valor. Esto conlleva la **transparencia referencial** (una función pura puede reemplazarse por su valor de retorno).
- **Ausencia de efectos colaterales:** Además del valor de retorno, una función pura no tiene ningún efecto visible sobre el ambiente desde el cual se la invoca.

→ Consecuencia práctica: Se hacen más fáciles las *Pruebas Unitarias*

Funciones puras

Ejemplo: función factorial (*!*)

```
Def inc ≡ + ∘ [id, 1̄]

Def aux ≡ > ∘ [1, 2] → 3; aux ∘ [inc ∘ 1, 2, apndr ∘ [3, 1]]

Def iota ≡ aux ∘ [1̄, id, ∅̄]

Def eq0 ≡ eq ∘ [id, 0̄]

Def ! ≡ eq0 → 1̄; (/×) ∘ iota

! : 5

120
```

Características de la Programación Funcional

3

DATOS INMUTABLES

- **Las funciones no modifican los datos recibidos como argumentos:** los valores retornados siempre están alojados en otras ubicaciones de memoria (las estructuras de datos son **persistentes**).

→ **Consecuencia práctica: Se hace más fácil la *Programación Concurrente***

En FP no existen las variables

Características de la Programación Funcional

4

FUNCIONES DE PRIMERA CLASE

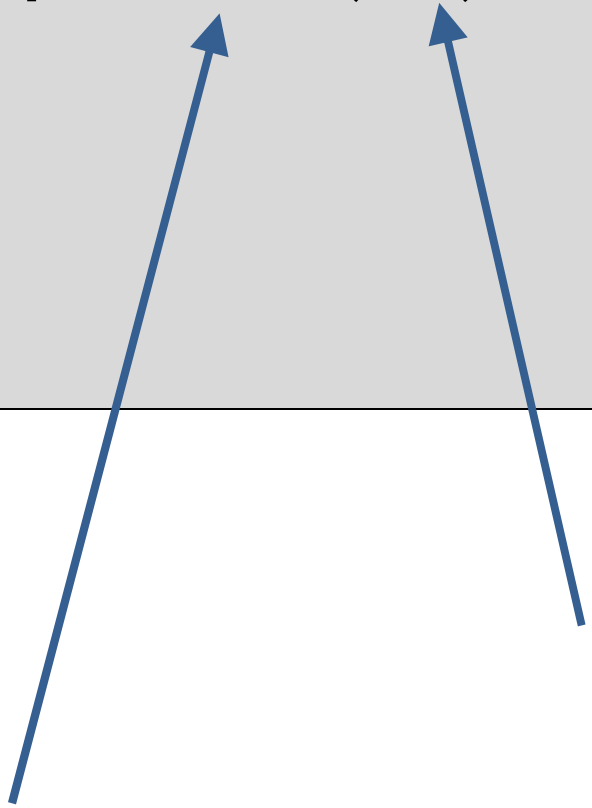
- **Soportan las operaciones posibles para las otras entidades del lenguaje:** pueden ser asignadas a una variable, ser pasadas como argumento y ser retornadas por una función.

Funciones de primera clase

```
Def ! ≡ eq0 → 1̄; (/×) ∘ iota

! : 5

120
```



la función × es recibida

la función 1̄ es retornada

Características de la Programación Funcional

5

FUNCIONES DE ORDEN SUPERIOR

Tienen alguna de las siguientes capacidades (o ambas):

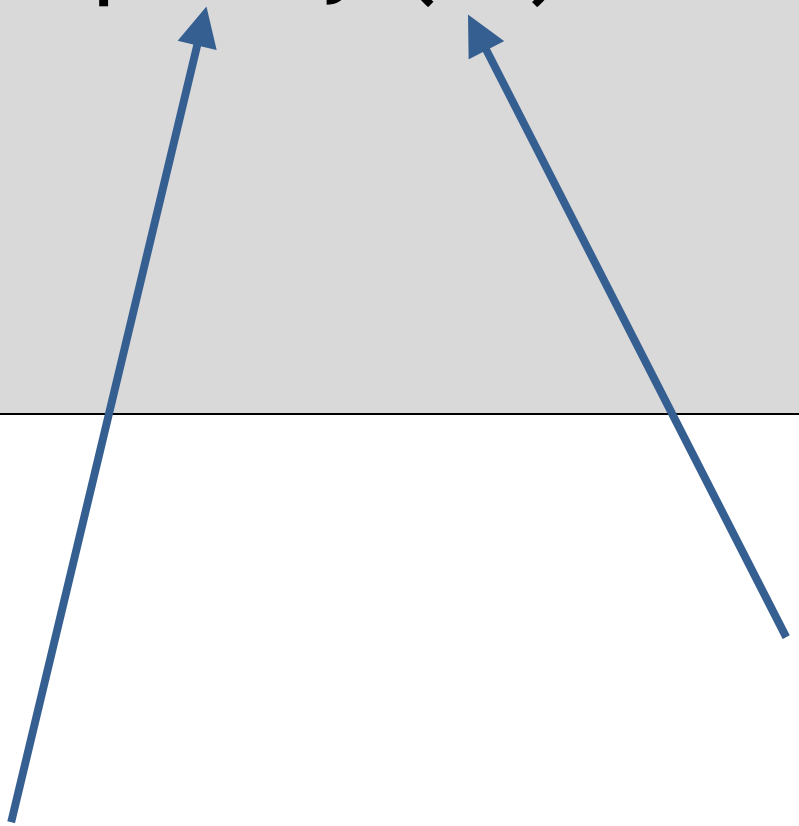
- **Recibir funciones como argumento**
- **Retornar una función**

Funciones de orden superior

```
Def ! ≡ eq0 → 1̄; (/×) ∘ iota

! : 5

120
```



/ recibe la función ×

→ retorna la función $\overline{1}$

Características de la Programación Funcional

6

COMPOSICIÓN DE FUNCIONES

Consiste en combinar funciones para formar otra:

- **El valor resultante se calcula aplicando una función a los argumentos. Luego se aplica otra al resultado, y así sucesivamente.**

Composición de funciones

Ejemplo: sumatoria de los recíprocos de los números de una colección

```
Def aux ≡ eq ∘ [1, 0̄] → 2; apndl
Def filtrar ≡ (/aux) ∘ apndr ∘ [id, ∅̄]
Def reciproco ≡ ÷ ∘ [1̄, id]
Def mapear ≡ α reciproco
Def reducir ≡ /+
Def sumrecipr ≡ reducir ∘ mapear ∘ filtrar
sumrecipr : <-1, 0, 2, 1>
0.5
```

Características de la Programación Funcional

7

RECURSIVIDAD

La inexistencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia y la iteración hacen que la repetición de instrucciones se implemente mediante funciones de orden superior o mediante funciones recursivas:

- **Una función recursiva es aquella que contiene, en el bloque de instrucciones que la definen, una llamada a la propia función, permitiendo que una operación se realice una y otra vez, hasta alcanzar el caso base.**

Recursividad

Ejemplo: función iota

```
Def inc ≡ + ∘ [id, 1̄]

Def aux ≡ > ∘ [1, 2] → 3; aux ∘ [inc ∘ 1, 2, apndr ∘ [3, 1]]

Def iota ≡ aux ∘ [1̄, id, ∅̄]

iota : 5

<1, 2, 3, 4, 5>
```