

# Paradigmas de Programación

## **Introducción a la materia Programación funcional básica**

**2do cuatrimestre de 2025**  
Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

Presentación de la materia

Tipos básicos y secuencias

Tipos de datos inductivos

Tipos abstractos de datos

Enumeraciones combinatorias

# Docentes (Turno Mañana)

## **Profesor**

- ▶ Christian Cossio-Mercado

## **Jefes de trabajos prácticos**

- ▶ Malena Ivnisky
- ▶ Brian Cardiff
- ▶ Daniel Grimaldi

## **Ayudantes de primera**

- ▶ Sebastián Felgueras

## **Ayudantes de segunda**

- ▶ Josefina Negrotto
- ▶ Facundo Chenlo
- ▶ Magali Kruel
- ▶ Pedro Arce Montoiro

## Días y horarios de cursada

- ▶ Martes de 9:00 a 14:00
- ▶ Viernes de 9:00 a 14:00

generalmente práctica

generalmente teórica

# Días y horarios de cursada

- ▶ Martes de 9:00 a 14:00 generalmente práctica
- ▶ Viernes de 9:00 a 14:00 generalmente teórica

## Consultas:

- ▶ Martes y viernes, cuando termine la clase ( 13:00/13:30 hs) hasta las 14 hs.

# Modalidad de evaluación

## Parciales

- |                                     |                        |
|-------------------------------------|------------------------|
| ▶ Primer parcial                    | martes 7 de octubre    |
| ▶ Segundo parcial                   | martes 25 de noviembre |
| ▶ Recuperatorio del primer parcial  | martes 2 de diciembre  |
| ▶ Recuperatorio del segundo parcial | martes 9 de diciembre  |

## Trabajos prácticos

- ▶ TP 1 (con su recuperatorio)
- ▶ TP 2 (con su recuperatorio)

Los TPs son en **grupos de 4 integrantes**.

## Examen final

(Con posibilidad de promoción).

# Materiales

Todo el material de la materia va a estar disponible en el campus:  
<https://campus.exactas.uba.ar/course/view.php?id=737>

- ▶ Diapositivas de las clases
- ▶ Guías de ejercicios
- ▶ Apuntes
- ▶ Enunciados de los trabajos prácticos
- ▶ Calendario
- ▶ ...

Revisen la sección “**útil**”.

# Vías de comunicación

## **Docentes → alumnxs**

Avisos a través del campus.

## **Alumnxs → docentes**

Lista de correo: `plp-docentes@dc.uba.ar`

(para consultas administrativas)

## **Discusión entre estudiantes fuera del horario de la materia**

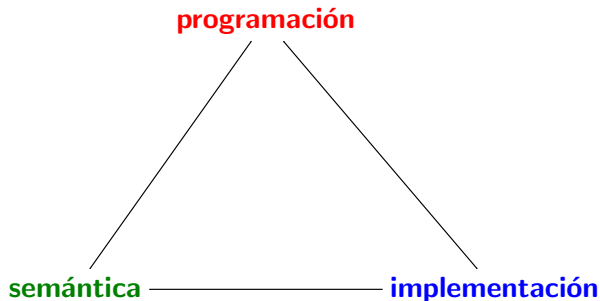
Servidor de Discord: `https://tinyurl.com/plpdiscord`

(con eventual participación de docentes)



# Contenidos

Tres aspectos de los **lenguajes de programación**:



# Cronograma

Programación funcional	2 semanas
Razonamiento ecuacional	1 semana
Lógica proposicional	1 semana
Cálculo- $\lambda$	2 semanas
(Repaso / consultas)	
<b>Primer parcial</b>	
Intérpretes e inferencia de tipos	1 semana
Unificación y lógica de primer orden	1 semana
Resolución	1 semana
Programación lógica	1,5 semanas
Programación orientada a objetos	1 semana
(Repaso / consultas)	
<b>Segundo parcial</b>	

# Motivación: programación

Los lenguajes de programación tienen distintas **características**.

- ▶ Etiquetado dinámico vs. tipado estático.
- ▶ Administración manual vs. automática de memoria.
- ▶ Funciones de primer orden vs. funciones de orden superior.
- ▶ Mutabilidad vs. inmutabilidad.
- ▶ Alcance dinámico vs. estático.
- ▶ Resolución de nombres temprana vs. tardía.
- ▶ Inferencia de tipos.
- ▶ Determinismo vs. no determinismo.
- ▶ Pasaje de parámetros por copia o por referencia.
- ▶ Evaluación estricta (por valor) o diferida (por necesidad).
- ▶ Tipos de datos inductivos, co-inductivos, GADTs, familias dependientes.
- ▶ *Pattern matching*, unificación.
- ▶ Polimorfismo paramétrico.
- ▶ Subclasificación, polimorfismo de subtipos, herencia simple vs. múltiple.
- ▶ Estructuras de control no local.
- ▶ ...

## Motivación: **programación**

Distintas características permiten abordar un mismo problema de distintas maneras.

$$\begin{array}{r} \text{CMXXIV} \\ + \quad \text{MCXLI} \\ \hline \end{array}$$

## Motivación: **programación**

Distintas características permiten abordar un mismo problema de distintas maneras.

$$\begin{array}{r} \text{CMXXIV} \\ + \text{MCXLI} \\ \hline \end{array}$$

$$\begin{array}{r} 924 \\ + 1141 \\ \hline \end{array}$$

## Motivación: programación

Distintas características permiten abordar un mismo problema de distintas maneras.

$$\begin{array}{r} \text{CMXXIV} \\ + \text{MCXLI} \\ \hline \end{array}$$

$$\begin{array}{r} 924 \\ + 1141 \\ \hline \end{array}$$

$$C = \{(x, y) \mid x^2 + y^2 = r^2\}$$

## Motivación: programación

Distintas características permiten abordar un mismo problema de distintas maneras.

$$\begin{array}{r} \text{CMXXIV} \\ + \text{MCXLI} \\ \hline \end{array}$$

$$\begin{array}{r} 924 \\ + 1141 \\ \hline \end{array}$$

$$C = \{(x, y) \mid x^2 + y^2 = r^2\} \quad C = \{(r \sin \theta, r \cos \theta) \mid 0 \leq \theta < 2\pi\}$$

## Motivación: programación

Distintas características permiten abordar un mismo problema de distintas maneras.

$$\begin{array}{r} \text{CMXXIV} \\ + \text{MCXLI} \\ \hline \end{array}$$

$$\begin{array}{r} 924 \\ + 1141 \\ \hline \end{array}$$

$$C = \{(x, y) \mid x^2 + y^2 = r^2\} \quad C = \{(r \sin \theta, r \cos \theta) \mid 0 \leq \theta < 2\pi\}$$

```
r := 1
while n > 0 {
  r := r * n
  n := n - 1
}
```



## Motivación: programación

Distintas características permiten abordar un mismo problema de distintas maneras.

$$\begin{array}{r} \text{CMXXIV} \\ + \text{MCXLI} \\ \hline \end{array}$$

$$\begin{array}{r} 924 \\ + 1141 \\ \hline \end{array}$$

$$C = \{(x, y) \mid x^2 + y^2 = r^2\} \quad C = \{(r \sin \theta, r \cos \theta) \mid 0 \leq \theta < 2\pi\}$$

```
r := 1
while n > 0 {
  r := r * n
  n := n - 1
}
```

```
foldl (*) 1 [1..n]
```

## Motivación: **semántica**

Dependemos del *software* en aplicaciones críticas.

- ▶ Telecomunicaciones.
- ▶ Procesos industriales.
- ▶ Reactores nucleares.
- ▶ Equipamiento médico.
- ▶ Previsión meteorológica.
- ▶ Aeronáutica.
- ▶ Vehículos autónomos.
- ▶ Transacciones monetarias.
- ▶ Análisis de datos en ciencia o toma de decisiones.
- ▶ ...

Las fallas cuestan recursos monetarios y vidas humanas.

## Motivación: **semántica**

¿Podemos confiar en que un programa hace lo que queremos?

## Motivación: **semántica**

- ¿Podemos confiar en que un programa hace lo que queremos?
- ¿Y si el programa está escrito por el enemigo?

## Motivación: **semántica**

¿Podemos confiar en que un programa hace lo que queremos?

¿Y si el programa está escrito por el enemigo?

¿Y si el programa está escrito por una IA?

## Motivación: **semántica**

¿Podemos confiar en que un programa hace lo que queremos?

¿Y si el programa está escrito por el enemigo?

¿Y si el programa está escrito por una IA?

### **Objetivo**

- ▶ Probar teoremas sobre el comportamiento de los programas.
- ▶ ¿Cómo darle significado matemático a los programas?

## Motivación: **semántica**

¿Podemos confiar en que un programa hace lo que queremos?

¿Y si el programa está escrito por el enemigo?

¿Y si el programa está escrito por una IA?

### **Objetivo**

- ▶ Probar teoremas sobre el comportamiento de los programas.
- ▶ ¿Cómo darle significado matemático a los programas?
- ▶ En AED vimos una manera de hacerlo (triplas de Hoare).
- ▶ En PLP veremos otras maneras de dar semántica.

## Motivación: **implementación**

Una computadora física ejecuta programas escritos en un lenguaje.  
(El “código máquina”).



## Motivación: **implementación**

Una computadora física ejecuta programas escritos en un lenguaje.  
(El “código máquina”).

¿Cómo es capaz de ejecutar programas escritos en otros lenguajes?

## Motivación: **implementación**

Una computadora física ejecuta programas escritos en un lenguaje.  
(El “código máquina”).

¿Cómo es capaz de ejecutar programas escritos en otros lenguajes?

- ▶ Interpretación (o evaluación).
- ▶ Chequeo e inferencia de tipos.
- ▶ Compilación (traducción de un lenguaje a otro).

## Bibliografía (no exhaustiva)

### **Lógica proposicional y de primer orden**

*Logic and Structure*

D. van Dalen.

### **Semántica y fundamentos de la implementación**

*Introduction to the Theory of Programming Languages*

J.-J. Lévy, G. Dowek. Springer, 2010.

*Types and Programming Languages*

B. Pierce. The MIT Press, 2002.

### **Programación funcional**

*Introduction to Functional Programming using Haskell*

R. Bird. Prentice Hall, 1998.

### **Programación lógica**

*Logic Programming with Prolog*

M. Bramer. Springer-Verlag, 2013.

### **Programación orientada a objetos**

*Smalltalk-80 the Language and its Implementation*

A. Goldberg, D. Robson. Addison-Wesley, 1983.

Presentación de la materia

Tipos básicos y secuencias

Tipos de datos inductivos

Tipos abstractos de datos

Enumeraciones combinatorias

# Programación con tipos básicos y secuencias

Definir las siguientes funciones:

- ▶ `factorial :: Int -> Int`  
dado un entero  $n \geq 0$ , devuelve  $n!$ .

# Programación con tipos básicos y secuencias

Definir las siguientes funciones:

- ▶ `factorial :: Int -> Int`  
dado un entero  $n \geq 0$ , devuelve  $n!$ .
- ▶ `sumaN :: Int -> [Int] -> [Int]`  
dado un entero  $k$  y una lista  $xs$ , devuelve la lista que resulta de sumarle  $k$  a cada elemento de  $xs$ .

# Programación con tipos básicos y secuencias

Definir las siguientes funciones:

- ▶ `factorial :: Int -> Int`  
dado un entero  $n \geq 0$ , devuelve  $n!$ .
- ▶ `sumaN :: Int -> [Int] -> [Int]`  
dado un entero  $k$  y una lista  $xs$ , devuelve la lista que resulta de sumarle  $k$  a cada elemento de  $xs$ .
- ▶ `aparece :: Char -> String -> Bool`  
dado un caracter  $c$  y un *string*  $s$ , devuelve un booleano que indica si  $c$  aparece en  $s$ .

# Programación con tipos básicos y secuencias

Definir las siguientes funciones:

- ▶ `factorial :: Int -> Int`  
dado un entero  $n \geq 0$ , devuelve  $n!$ .
- ▶ `sumaN :: Int -> [Int] -> [Int]`  
dado un entero  $k$  y una lista  $xs$ , devuelve la lista que resulta de sumarle  $k$  a cada elemento de  $xs$ .
- ▶ `aparece :: Char -> String -> Bool`  
dado un caracter  $c$  y un *string*  $s$ , devuelve un booleano que indica si  $c$  aparece en  $s$ .

Más en general:

`aparece :: Eq a => a -> [a] -> Bool`



# Programación con tipos básicos y secuencias

Definir las siguientes funciones:

- ▶ `factorial :: Int -> Int`  
dado un entero  $n \geq 0$ , devuelve  $n!$ .
- ▶ `sumaN :: Int -> [Int] -> [Int]`  
dado un entero  $k$  y una lista  $xs$ , devuelve la lista que resulta de sumarle  $k$  a cada elemento de  $xs$ .
- ▶ `aparece :: Char -> String -> Bool`  
dado un caracter  $c$  y un *string*  $s$ , devuelve un booleano que indica si  $c$  aparece en  $s$ .

Más en general:

- `aparece :: Eq a => a -> [a] -> Bool`
- ▶ `ordenar :: [Float] -> [Float]`  
dada una lista, devuelve su permutación ordenada.

# Programación con tipos básicos y secuencias

Definir las siguientes funciones:

- ▶ `factorial :: Int -> Int`  
dado un entero  $n \geq 0$ , devuelve  $n!$ .
- ▶ `sumaN :: Int -> [Int] -> [Int]`  
dado un entero  $k$  y una lista  $xs$ , devuelve la lista que resulta de sumarle  $k$  a cada elemento de  $xs$ .
- ▶ `aparece :: Char -> String -> Bool`  
dado un caracter  $c$  y un *string*  $s$ , devuelve un booleano que indica si  $c$  aparece en  $s$ .

Más en general:

`aparece :: Eq a => a -> [a] -> Bool`

- ▶ `ordenar :: [Float] -> [Float]`  
dada una lista, devuelve su permutación ordenada.

Más en general:

`ordenar :: Ord a => [a] -> [a]`

Presentación de la materia

Tipos básicos y secuencias

**Tipos de datos inductivos**

Tipos abstractos de datos

Enumeraciones combinatorias

# Tipos enumerados

Dado el siguiente tipo de datos:

```
data Direccion = Norte | Este | Sur | Oeste
```

definir la función

```
opuesta :: Direccion -> Direccion
```

que dada una dirección  $d$ , devuelve la dirección opuesta a  $d$ .

## Tipos opcionales

Definir la función

`últimoÍndiceDe :: Eq a => a -> [a] -> Int`

que dado un elemento  $x$  y una lista de elementos  $xs$ , devuelve el índice de la *última* ocurrencia de  $x$  en  $xs$ .

## Tipos opcionales

Definir la función

`últimoÍndiceDe :: Eq a => a -> [a] -> Int`

que dado un elemento  $x$  y una lista de elementos  $xs$ , devuelve el índice de la *última* ocurrencia de  $x$  en  $xs$ .

Es una función parcial.

## Tipos opcionales

Definir la función

`últimoÍndiceDe :: Eq a => a -> [a] -> Int`

que dado un elemento  $x$  y una lista de elementos  $xs$ , devuelve el índice de la *última* ocurrencia de  $x$  en  $xs$ .

Es una función parcial. ¿Cómo la podemos hacer total?

# Tipos opcionales

Definir la función

```
últimoÍndiceDe :: Eq a => a -> [a] -> Int
```

que dado un elemento  $x$  y una lista de elementos  $xs$ , devuelve el índice de la *última* ocurrencia de  $x$  en  $xs$ .

Es una función parcial. ¿Cómo la podemos hacer total?

Podemos usar el siguiente tipo de datos:

```
data Maybe a = Nothing | Just a
```



# Tipos opcionales

Definir la función

```
últimoÍndiceDe :: Eq a => a -> [a] -> Int
```

que dado un elemento  $x$  y una lista de elementos  $xs$ , devuelve el índice de la *última* ocurrencia de  $x$  en  $xs$ .

Es una función parcial. ¿Cómo la podemos hacer total?

Podemos usar el siguiente tipo de datos:

```
data Maybe a = Nothing | Just a
```

Redefinir ahora la función para que sea total, con el siguiente tipo:

```
últimoÍndiceDe :: Eq a => a -> [a] -> Maybe Int
```

# Árboles

Dado el siguiente tipo de datos:

```
data AB a = Nil | Bin (AB a) a (AB a)
```

# Árboles

Dado el siguiente tipo de datos:

```
data AB a = Nil | Bin (AB a) a (AB a)
```

- Dibujar y escribir en Haskell todos los árboles que tienen 3 nodos, en todos los cuales se encuentra el número 0.

# Árboles

Dado el siguiente tipo de datos:

```
data AB a = Nil | Bin (AB a) a (AB a)
```

- ▶ Dibujar y escribir en Haskell todos los árboles que tienen 3 nodos, en todos los cuales se encuentra el número 0.
- ▶ Definir las funciones:
  1. `preorder :: AB a -> [a]`
  2. `inorder :: AB a -> [a]`
  3. `postorder :: AB a -> [a]`

Presentación de la materia

Tipos básicos y secuencias

Tipos de datos inductivos

**Tipos abstractos de datos**

Enumeraciones combinatorias

## Conjunto sobre listas

Implementemos un **conjunto** con la siguiente interfaz:

```
vacío      :: Conj a
insertar   :: Eq a => a -> Conj a -> Conj a
pertenece  :: Eq a => a -> Conj a -> Bool
eliminar   :: Eq a => a -> Conj a -> Conj a
```

## Conjunto sobre listas

Implementemos un **conjunto** con la siguiente interfaz:

```
vacío      :: Conj a
insertar   :: Eq a => a -> Conj a -> Conj a
pertenece  :: Eq a => a -> Conj a -> Bool
eliminar   :: Eq a => a -> Conj a -> Conj a
```

Elegimos la siguiente estructura de representación:

```
data Conj a = CConj [a]
```

## Conjunto sobre listas

Implementemos un **conjunto** con la siguiente interfaz:

```
vacío      :: Conj a
insertar   :: Eq a => a -> Conj a -> Conj a
pertenece  :: Eq a => a -> Conj a -> Bool
eliminar   :: Eq a => a -> Conj a -> Conj a
```

Elegimos la siguiente estructura de representación:

```
data Conj a = CConj [a]
```

con el siguiente invariante:

- La lista no debe contener elementos repetidos.



# Diccionario sobre árboles binarios de búsqueda

Implementemos un **diccionario** con la siguiente interfaz:

```
vacío    :: Dict k v
definir  :: Ord k => k -> v -> Dict k v -> Dict k v
buscar   :: Ord k => k -> Dict k v -> Maybe v
```

# Diccionario sobre árboles binarios de búsqueda

Implementemos un **diccionario** con la siguiente interfaz:

```
vacío    :: Dict k v
definir  :: Ord k => k -> v -> Dict k v -> Dict k v
buscar   :: Ord k => k -> Dict k v -> Maybe v
```

Elegimos la siguiente estructura de representación:

```
data Dict k v = CDict (AB (k, v))
```

# Diccionario sobre árboles binarios de búsqueda

Implementemos un **diccionario** con la siguiente interfaz:

```
vacío    :: Dict k v
definir  :: Ord k => k -> v -> Dict k v -> Dict k v
buscar   :: Ord k => k -> Dict k v -> Maybe v
```

Elegimos la siguiente estructura de representación:

```
data Dict k v = CDict (AB (k, v))
```

con el siguiente invariante:

- ▶ El árbol binario debe ser un **árbol binario de búsqueda**.  
Es decir, en cada subárbol:
  - ▶ Las claves del subárbol izquierdo son menores que la raíz.
  - ▶ Las claves del subárbol derecho son mayores que la raíz.

Presentación de la materia

Tipos básicos y secuencias

Tipos de datos inductivos

Tipos abstractos de datos

Enumeraciones combinatorias

# Subsecuencias

Definir una función: `subsecuencias :: [a] -> [[a]]` que dada una lista, devuelva la lista de todas sus posibles *subsecuencias*.

# Subsecuencias

Definir una función: `subsecuencias :: [a] -> [[a]]` que dada una lista, devuelva la lista de todas sus posibles *subsecuencias*.

Por ejemplo, las subsecuencias de `[1, 2, 3]` son:

`[[], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]]`

# Permutaciones

Definir una función: `permutaciones :: [a] -> [[a]]` que dada una lista, devuelva la lista de todas sus posibles *permutaciones*.

# Permutaciones

Definir una función: `permutaciones :: [a] -> [[a]]` que dada una lista, devuelva la lista de todas sus posibles *permutaciones*.

Por ejemplo, las permutaciones de `[1, 2, 3]` son:

```
[1, 2, 3], [1, 3, 2], [2, 1, 3]  
[2, 3, 1], [3, 1, 2], [3, 2, 1]]
```



ι ι ι ι ι ι ι ι ι ι ? ? ? ? ? ? ? ?

Lectura recomendada

**Capítulos 1–3 del libro de Bird.**

Richard Bird. *Thinking functionally with Haskell*.

Cambridge University Press, 2015.