# Lenguajes de Programación I

## Programación Funcional

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"







## Programación Funcional

### Conceptos Fundamentales

- **Programación Funcional**
  - Salida de un programa es una función de sus entradas
  - Se excluyen las nociones de estado mutable y efecto de borde.
- Derivada del $\lambda$-**Cálculo** de Alonso Church (1936) – equivalente al modelo imperativo (Tesis de Church-Turing).
- Caracterizada por:
  - Funciones de primera clase.
  - Funciones de orden superior.
  - Funciones que retornan valores estructurados.
  - Polimorfismo dinámico (Racket) o estático (Haskell).
  - Tipo Lista con abundantes operadores.
  - Recursión.
  - Tipos agregados definibles por el usuario.
  - Recolección de basura.





## Programar es evaluar

### Transformar valores por aplicación de funciones

- Programar funcionalmente requiere:
  - Definir funciones.
  - Componerlas.
  - Aplicarlas o evaluarlas.
- Ciclo *read-eval-loop* en lenguajes interpretados – Racket.
- Compilar y ejecutar en lenguajes compilados – Haskell.





## Lenguajes Interpretados

### LISP y sus dialectos... Racket es el favorito

- El interpretador lee **expresiones-S** (*S-expressions*).
- La expresión se evalúa
  - Notación polaca prefija.
  - Paréntesis indican aplicación funcional.
  - Primer elemento de la expresión se asume como función.
  - Resto de los elementos se asumen como argumentos.
- Manejo dinámico de tipos.
  - Números.
  - Símbolos (también llamados *átomos*).
  - Listas.
  - Cadenas – sólo en implantaciones modernas.
- Funciones anónimas – *lambda expressions*.

## Asociaciones

- Asociar nombres a valores usando un alcance estático.
  - Lista de asociaciones paralelas (`let`).
  - Lista de asociaciones ordenadas (`let*`).
  - Lista de asociaciones recursivas (`letrec`).
  - La intención es simplificar expresiones.
- Asociaciones *puras* – valores no pueden modificarse en las expresiones.
- Asociaciones globales (`define`) – para definir funciones con nombre.





## Tipos de Datos

- Listas heterogéneas.
  - Funciones para descomposición (`car`, `cdr` y sus extensiones).
  - Funciones para composición (`cons`, `append` y `list`)
  - Predicados (`null?` para vacuidad).
  - Listas de Asociación (`assq` para búsqueda).
- Números.
  - Funciones aritméticas y trascendentales.
  - Predicados para comparación.
- Booleanos (`#t` y `#f`):
- Comparación.
  - ¿Son el mismo objeto? (`eq?`).
  - ¿Son semánticamente equivalentes? (`eqv?`).
  - ¿Son estructuralmente equivalentes? (`equal?`).





## Estructuras de Control

- Selección Simple (`if`).
- Selección Generalizada (`cond`).
- Recursión.
- Los programas vistos como listas
  - Evaluación de expresiones (`eval`).
  - Aplicación de funciones (`apply`).





## Tratamiento avanzado

- I/O – rompe la pureza del lenguaje en Racket.
  - Interactivo o con archivos (*ports*).
  - `read`/`write` vs. `read-char`/`write-char`.
- Vectores vs. Listas.
- Macros "higiénicos" – *garantizan* que no habrá colisiones.
  - Si definen nombres, *nunca* coincidirán con nombres existentes.
  - Si definen nombres *libres*, refieren al ambiente previo a la expansión.
- Continuaciones – `call-cc`
  - Estructuras de control *ad hoc*.
  - Excepciones.
  - Co-rutinas.
- Evaluación diferida (`delay` y `force`).

# Orden de Evaluación

- **Aplicativo** – evaluar argumentos antes de evaluar la función.
- **Normal** – evaluar la función antes que los argumentos, y éstos últimos se evalúan solamente si son necesarios.
- Ambos órdenes llegan al mismo valor final, si está definido.
- Ninguna es particularmente más eficiente que la otra.





# Evaluación Perezosa

- **Función Estricta**
  - *Requiere* que todos sus argumentos estén definidos.
  - Obtener el valor resultado no depende del orden de evaluación – si los argumentos están definidos, siempre puede calcularse.
- **Función No Estricta**
  - Puede operar con argumentos que no están definidos.
  - Calcular el resultado depende de que sólo se evalúen argumentos definidos en el orden preciso.
- Un lenguaje es estricto si obliga funciones estrictas – Racket, ML.
- Un lenguaje no es estricto en el caso contrario – Haskell, R.
- La Evaluación Perezosa combina
  - Orden Normal de Evaluación.
  - Ejecución a velocidad equivalente a lenguajes estrictos.
  - Valores se marcan como "promesa" evaluada sólo si es necesaria.
  - Particularmente útil para esctructuras infinitas.





# Funciones de Orden Superior

- Una función es de **Orden Superior** si recibe funciones como argumentos o retorna funciones como resultado.
- También se les llama **Formas Funcionales**.
- Aplicación explícita.
- Composición de funciones.
- Iteración implícita
  - `map`
  - `filter`
  - `folds`
  - `scans`
- Currying y Un-currying.





# ¡Pero el mundo real no es puro!

**Es un gran efecto de borde**

- Los primeros lenguajes funcionales no tuvieron reparos en incluir componentes impuros para ser útiles
  - `read` y `display` para leer y escribir valores.
  - `set!` para modificar valores en sitio.
- ¿Y si imaginamos al mundo como un "fluído"?
  - Una lista infinita de entradas de la cual tomar datos en orden...
  - ...y poner resultados en una infinita lista de salida.
  - *Streams* de ML y las primeras versiones de Haskell.
  - No modelan correctamente las interfaces modales por eventos, el acceso directo a archivos, ni el polimorfismo.
- ¿Y si tuviéramos el mundo en la mano? – un valor de estado mutable
  - Funciones de orden superior que se aplican en un orden específico – un "punto y coma" controlable.
  - Programador define *Monads* concretas para sus propósitos – Haskell provee un *Monad* abstracto para IO.

# Bibliografía


- [*Scott*]
  - Capítulo 10
  - Ejercicios y Exploraciones
- The Scheme Programming Language
- Structure and Interpretation of Computer Programs