# Paradigmas Fundamentales de Programación

## Introducción al modelo declarativo

Juan Francisco Díaz Frias

Maestría en Ingeniería, Énfasis en Ingeniería de Sistemas y Computación
Escuela de Ingeniería de Sistemas y Computación,
home page: http://eisc.univalle.edu.co
Universidad del Valle - Cali, Colombia

# Modelos y Paradigmas de Programación

Universidad del Valle

## Plan

# Plan

# Plan

# ¿Qué entendemos por programación?

La programación abarca tres cosas:

- Modelo de computación:sistema formal que define un lenguaje y cómo se ejecutan las frases del lenguaje por parte de una máquina abstracta.
- Modelo de programación: conjunto de técnicas de programación y principios de diseño utilizados para escribir programas en el lenguaje del modelo de computación.
- Técnicas de razonamiento: corrección y eficiencia.

¿Qué tipo de modelos usar?

Aquellos que se puedan usar para resolver muchos problemas, que cuenten con técnicas de razonamiento prácticas y sencillas, y que se puedan implementar eficientemente.

# ¿Qué entendemos por programación?

La programación abarca tres cosas:

- Modelo de computación:sistema formal que define un lenguaje y cómo se ejecutan las frases del lenguaje por parte de una máquina abstracta.
- Modelo de programación: conjunto de técnicas de programación y principios de diseño utilizados para escribir programas en el lenguaje del modelo de computación.
- Técnicas de razonamiento: corrección y eficiencia.

¿Qué tipo de modelos usar?

Aquellos que se puedan usar para resolver muchos problemas, que cuenten con técnicas de razonamiento prácticas y sencillas, y que se puedan implementar eficientemente.

# ¿Qué entendemos por programación?

La programación abarca tres cosas:

- Modelo de computación:sistema formal que define un lenguaje y cómo se ejecutan las frases del lenguaje por parte de una máquina abstracta.
- Modelo de programación: conjunto de técnicas de programación y principios de diseño utilizados para escribir programas en el lenguaje del modelo de computación.
- Técnicas de razonamiento: corrección y eficiencia.

¿Qué tipo de modelos usar?

Aquellos que se puedan usar para resolver muchos problemas, que cuenten con técnicas de razonamiento prácticas y sencillas, y que se puedan implementar eficientemente.

# ¿Qué entendemos por programación?

La programación abarca tres cosas:

- Modelo de computación:sistema formal que define un lenguaje y cómo se ejecutan las frases del lenguaje por parte de una máquina abstracta.
- Modelo de programación: conjunto de técnicas de programación y principios de diseño utilizados para escribir programas en el lenguaje del modelo de computación.
- Técnicas de razonamiento: corrección y eficiencia.

## ¿Qué tipo de modelos usar?

Aquellos que se puedan usar para resolver muchos problemas, que cuenten con técnicas de razonamiento prácticas y sencillas, y que se puedan implementar eficientemente.

# El modelo de programación declarativo

- Evaluación de funciones sobre estructuras de datos parciales.
- Abarca los principales paradigmas declarativos: programación funcional (LISP, Scheme, ML, Haskell) y lógica (Prolog, Mercury).
- Programación sin estado.
- Subyace la POO.

# El modelo de programación declarativo

- Evaluación de funciones sobre estructuras de datos parciales.
- Abarca los principales paradigmas declarativos: programación funcional (LISP, Scheme, ML, Haskell) y lógica (Prolog, Mercury).
- Programación sin estado.
- Subyace la POO.

# El modelo de programación declarativo

- Evaluación de funciones sobre estructuras de datos parciales.
- Abarca los principales paradigmas declarativos: programación funcional (LISP, Scheme, ML, Haskell) y lógica (Prolog, Mercury).
- Programación sin estado.
- Subyace la POO.

# El modelo de programación declarativo

- Evaluación de funciones sobre estructuras de datos parciales.
- Abarca los principales paradigmas declarativos: programación funcional (LISP, Scheme, ML, Haskell) y lógica (Prolog, Mercury).
- Programación sin estado.
- Subyace la POO.

# Lenguajes de programación prácticos

- Son los lenguajes usados para resolver problemas del mundo real.
- Un lenguaje práctico es como la caja de herramientas de un mecánico experimentado: hay muchas herramientas diferentes para muchos propósitos distintos y todas tienen una razón para estar allí.
- Recordaremos las técnicas para presentar la sintaxis (“gramática”) y la semántica (“significado”) de los lenguajes de programación prácticos.

# Lenguajes de programación prácticos

- Son los lenguajes usados para resolver problemas del mundo real.
- Un lenguaje práctico es como la caja de herramientas de un mecánico experimentado: hay muchas herramientas diferentes para muchos propósitos distintos y todas tienen una razón para estar allí.
- Recordaremos las técnicas para presentar la sintaxis (“gramática”) y la semántica (“significado”) de los lenguajes de programación prácticos.

# Lenguajes de programación prácticos

- Son los lenguajes usados para resolver problemas del mundo real.
- Un lenguaje práctico es como la caja de herramientas de un mecánico experimentado: hay muchas herramientas diferentes para muchos propósitos distintos y todas tienen una razón para estar allí.
- Recordaremos las técnicas para presentar la sintaxis (“gramática”) y la semántica (“significado”) de los lenguajes de programación prácticos.

# Plan

# Sintaxis (1)

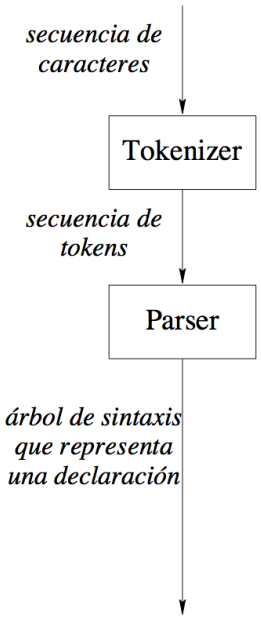
- La sintaxis de un lenguaje define cómo son los programas legales, no lo que hacen.
- Gramática: reglas que definen cómo construir 'frases'(declaraciones) a partir de 'palabras'(lexemas o *tokens*).
- En los LP:

  declaración = sec. de lexemas
  lexema = sec. de caracs.

- Ejemplo:

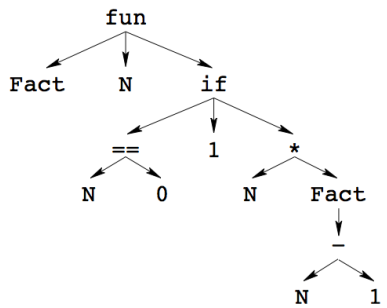
```
fun {Fact N}
   if N==0 then 1
   else N*{Fact N-1}
   end
end
```

## Análisis léxico y sintáctico




```
[f u n '{' 'F' a c t ' ' 'N' '}' '\n' ' ' i f ' '
 'N' '=' '=' 0 ' ' t h e n ' ' 1 '\n' ' ' e l s e
 ' ' 'N' '*' '{' 'F' a c t ' ' 'N' '-' 1 '}' ' ' e n
 d '\n' e n d]
```

```
['fun' '{' 'Fact' 'N' '}' 'if' 'N' '==' '0' 'then'
 1 'else' 'N' '*' '{' 'Fact' 'N' '-' '1' '}' 'end'
 'end']
```



# Sintaxis (1)

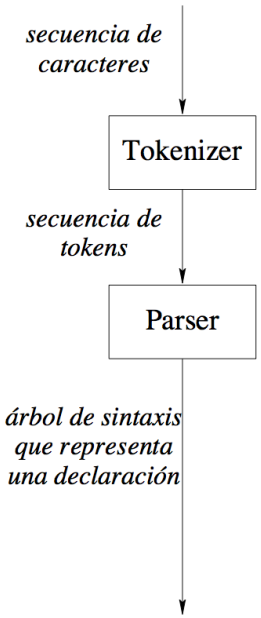
- La sintaxis de un lenguaje define cómo son los programas legales, no lo que hacen.
- Gramática: reglas que definen cómo construir 'frases'(declaraciones) a partir de 'palabras'(lexemas o *tokens*).
- En los LP:

  declaración = sec. de lexemas
  lexema = sec. de caracs.

- Ejemplo:

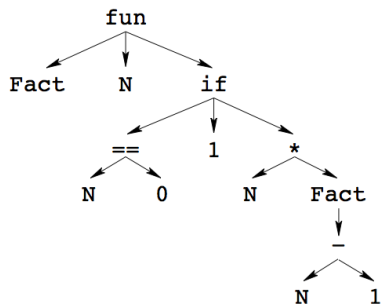
```
fun {Fact N}
   if N==0 then 1
   else N*{Fact N-1}
   end
end
```

## Análisis léxico y sintáctico




```
[f u n '{' 'F' a c t ' ' 'N' '}' '\n' ' ' i f ' '
 'N' '=' '=' 0 ' ' t h e n ' ' 1 '\n' ' ' e l s e
 ' ' 'N' '*' '{' 'F' a c t ' ' 'N' '-' 1 '}' ' ' e n
 d '\n' e n d]
```

```
['fun' '{' 'Fact' 'N' '}' 'if' 'N' '==' '0' 'then'
 1 'else' 'N' '*' '{' 'Fact' 'N' '-' '1' '}' 'end'
 'end']
```



# Sintaxis (1)

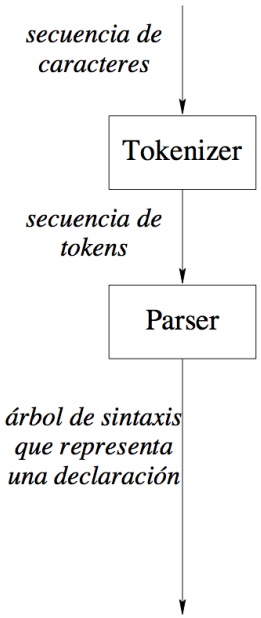
- La sintaxis de un lenguaje define cómo son los programas legales, no lo que hacen.
- Gramática: reglas que definen cómo construir 'frases'(declaraciones) a partir de 'palabras'(lexemas o *tokens*).
- En los LP:

| | | |
|---|---|---|
| declaración | = | sec. de lexemas |
| lexema | = | sec. de caracs. |

- Ejemplo:

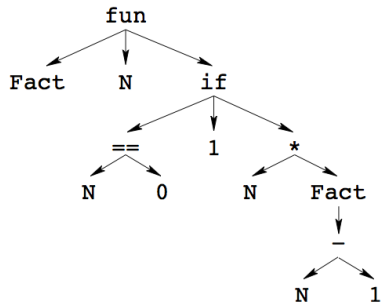
```
fun {Fact N}
   if N==0 then 1
   else N*{Fact N-1}
   end
end
```

## Análisis léxico y sintáctico




```
[f u n '{' 'F' a c t ' ' 'N' '}' '\n' ' ' i f ' '
 'N' '=' '=' 0 ' ' t h e n ' ' 1 '\n' ' ' e l s e
 ' ' 'N' '*' '{' 'F' a c t ' ' 'N' '-' 1 '}' ' ' e n
 d '\n' e n d]
```

```
['fun' '{' 'Fact' 'N' '}' 'if' 'N' '==' '0' 'then'
 1 'else' 'N' '*' '{' 'Fact' 'N' '-' '1' '}' 'end'
 'end']
```



# Sintaxis (1)

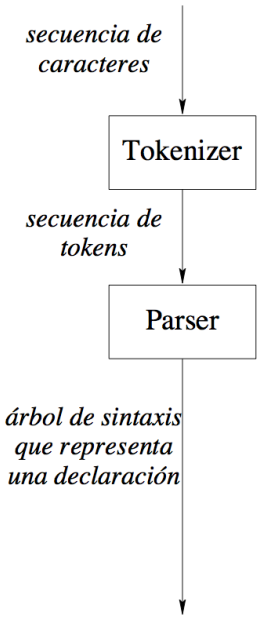
- La sintaxis de un lenguaje define cómo son los programas legales, no lo que hacen.
- Gramática: reglas que definen cómo construir 'frases'(declaraciones) a partir de 'palabras'(lexemas o *tokens*).
- En los LP:

| | | |
|---|---|---|
| declaración | = | sec. de lexemas |
| lexema | = | sec. de caracs. |

- Ejemplo:

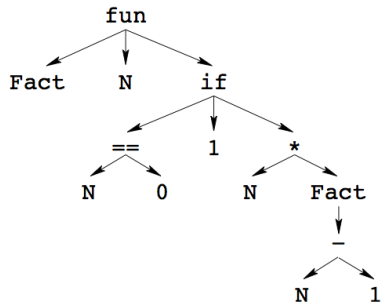
```
fun {Fact N}
 if N==0 then 1
 else N*{Fact N-1}
 end
end
```

## Análisis léxico y sintáctico




```
[f u n '{' 'F' a c t ' ' 'N' '}' '\n' ' ' i f ' '
 'N' '=' '=' 0 ' ' t h e n ' ' 1 '\n' ' ' e l s e
 ' ' 'N' '*' '{' 'F' a c t ' ' 'N' '-' 1 '}' ' ' e n
 d '\n' e n d]
```

```
['fun' '{' 'Fact' 'N' '}' 'if' 'N' '==' '0' 'then'
 1 'else' 'N' '*' '{' 'Fact' 'N' '-' '1' '}' 'end'
 'end']
```



# Sintaxis (2)

## La notación de Backus-Naur extendida (EBNF)

- EBNF distingue entre símbolos terminales y símbolos no-terminales.
- Símbolo terminal:*lexema*.
- Símbolo no-terminal:secuencia de *lexemas*.
- Ejemplo: $\langle digito \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
  $\langle ent \rangle ::= \langle digito \rangle \{ \langle digito \rangle \}$

# Sintaxis (2)

## La notación de Backus-Naur extendida (EBNF)

- EBNF distingue entre símbolos terminales y símbolos no-terminales.
- Símbolo terminal:*lexema*.
- Símbolo no-terminal:secuencia de *lexemas*.
- Ejemplo: $\langle digito \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
  $\langle ent \rangle ::= \langle digito \rangle \{ \langle digito \rangle \}$

# Sintaxis (2)

## La notación de Backus-Naur extendida (EBNF)

- EBNF distingue entre símbolos terminales y símbolos no-terminales.
- Símbolo terminal:*lexema*.
- Símbolo no-terminal:secuencia de *lexemas*.
- Ejemplo: $\langle digito \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
  $\langle ent \rangle ::= \langle digito \rangle \{ \langle digito \rangle \}$

# Sintaxis (2)

## La notación de Backus-Naur extendida (EBNF)

- EBNF distingue entre símbolos terminales y símbolos no-terminales.
- Símbolo terminal:*lexema*.
- Símbolo no-terminal:secuencia de *lexemas*.
- Ejemplo:
$$\begin{aligned} \langle \text{digito} \rangle &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ \langle \text{ent} \rangle &::= \langle \text{digito} \rangle \{ \langle \text{digito} \rangle \} \end{aligned}$$

# Sintaxis (3)

## Expresividad de EBNF

- EBNF $\equiv$ gramáticas independientes del contexto.
- Una gramática EBNF no puede espresar que una variable deba ser declarada antes de ser usada.
- Por ello, la sintaxis se describe:

Gramática independiente del contexto (e.g., con EBNF)

+

Conjunto de condiciones adicionales

– *Es fácil de leer y entender*

– *Define un superconjunto del lenguaje*

– *Expresa las restricciones impuestas por el lenguaje (e.g., la declaración de las variables debe hacerse antes de usarlas)*

– *La gramática se vuelve sensible al contexto*

# Sintaxis (4)

## Ejemplos de reglas

- ⟨declaración⟩ ::= `skip` | ⟨expresión⟩ `'='` ⟨expresión⟩ | ...
  ⟨expresión⟩ ::= ⟨variable⟩ | ⟨ent⟩ | ...
- ⟨declaración⟩ ::= `if` ⟨expresión⟩ `then` ⟨declaración⟩
  { `elseif` ⟨expresión⟩ `then` ⟨declaración⟩ }
  [ `else` ⟨declaración⟩ ] `end` | ...
- ⟨expresión⟩ ::= `'['` { ⟨expresión⟩ }+ `']'` | ...
  ⟨etiqueta⟩ ::= `unit` | `true` | `false` | ⟨variable⟩ | ⟨átomo⟩

# Sintaxis (4)

## Ejemplos de reglas

- ⟨declaración⟩ ::= **skip** | ⟨expresión⟩ '=' ⟨expresión⟩ | ...
  ⟨expresión⟩ ::= ⟨variable⟩ | ⟨ent⟩ | ...
- ⟨declaración⟩ ::= **if** ⟨expresión⟩ **then** ⟨declaración⟩
  { **elseif** ⟨expresión⟩ **then** ⟨declaración⟩ }
  [ **else** ⟨declaración⟩ ] **end** | ...
- ⟨expresión⟩ ::= '[' { ⟨expresión⟩ }+ ']' | ...
  ⟨etiqueta⟩ ::= **unit** | **true** | **false** | ⟨variable⟩ | ⟨átomo⟩

# Plan

# Semántica (1)

- La semántica de un lenguaje define lo que hace un programa cuando se ejecuta.
- La semántica debería ser simple y permitir razonar sobre el programa (corrección, tiempo de ejecución, y utilización de memoria).
- ¿Se puede lograr esto para un lenguaje de programación práctico?
- Enfoque del lenguaje núcleo

## Semántica (1)

- La semántica de un lenguaje define lo que hace un programa cuando se ejecuta.
- La semántica debería ser simple y permitir razonar sobre el programa (corrección, tiempo de ejecución, y utilización de memoria).
- ¿Se puede lograr esto para un lenguaje de programación práctico?
- Enfoque del lenguaje núcleo

# Semántica (1)

- La semántica de un lenguaje define lo que hace un programa cuando se ejecuta.
- La semántica debería ser simple y permitir razonar sobre el programa (corrección, tiempo de ejecución, y utilización de memoria).
- ¿Se puede lograr esto para un lenguaje de programación práctico?
- Enfoque del lenguaje núcleo

## Semántica (1)

- La semántica de un lenguaje define lo que hace un programa cuando se ejecuta.
- La semántica debería ser simple y permitir razonar sobre el programa (corrección, tiempo de ejecución, y utilización de memoria).
- ¿Se puede lograr esto para un lenguaje de programación práctico?
- Enfoque del lenguaje núcleo

## El enfoque del lenguaje núcleo (1)

Todas las construcciones del lenguaje se definen en términos de traducciones a un lenguaje básico, conocido como el lenguaje núcleo. Dos partes:

- Definición del lenguaje núcleo y de las estructuras de datos que manipula.
- Esquema de traducción de todo el lenguaje de programación al lenguaje núcleo. Dos clases de traducciones:
  - Abstracciones lingüísticas.
  - Azúcar sintáctico.

# Semántica (2)

## El enfoque del lenguaje núcleo (1)

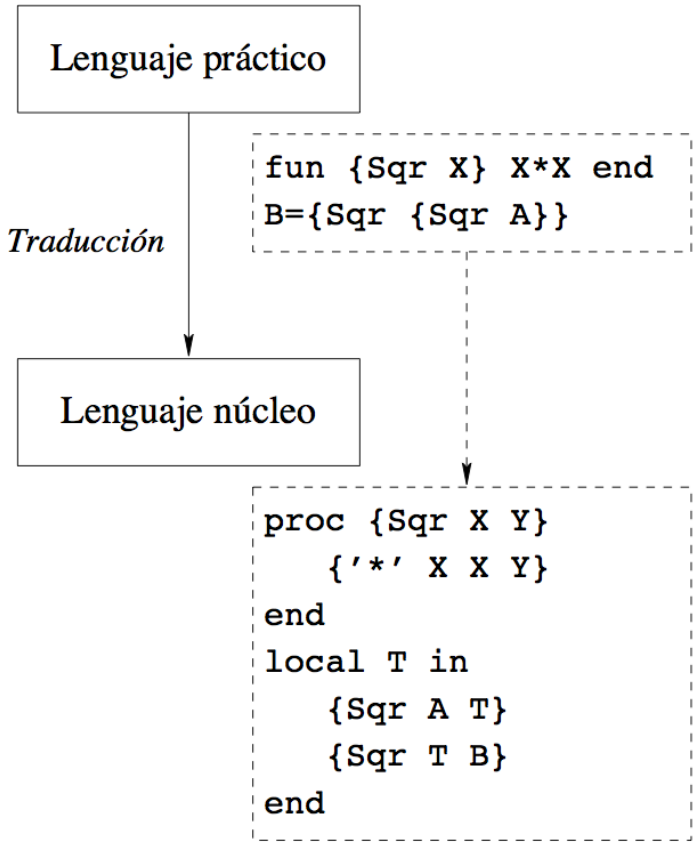
Todas las construcciones del lenguaje se definen en términos de traducciones a un lenguaje básico, conocido como el lenguaje núcleo. Dos partes:

- Definición del lenguaje núcleo y de las estructuras de datos que manipula.
- Esquema de traducción de todo el lenguaje de programación al lenguaje núcleo. Dos clases de traducciones:
  - Abstracciones lingüísticas.
  - Azúcar sintáctico.

Universidad del Valle

# Semántica (3)

## El enfoque del lenguaje núcleo (2)




- *Provee abstracciones útiles para el programador*
- *Se puede extender con abstracciones lingüísticas*
- *Contiene un conjunto minimal de conceptos intuitivos*
- *Para el programador es fácil de entender y razonar con él*
- *Está definido por una semántica formal (operacional, axiomática o denotacional)*

# Semántica (4)

## Cuatro enfoques de semántica de LP

- Una semántica axiomática: la semántica de una declaración es la relación entre el estado de entrada (la situación antes de ejecutarse la declaración) y el estado de salida (la situación despues de ejecutarse la declaración). Funciona bien en modelos con estado.
- Una semántica denotacional: una declaración es una función sobre un dominio abstracto. Funciona bien para modelos declarativos, pero es complicado cuando se aplica a lenguajes concurrentes.
- Una semántica lógica: una declaración es un modelo de una teoría lógica. Funciona bien para los modelos de computación declarativa y relacional.
- Una semántica operacional: ejecución de una declaración en términos de una máquina abstracta. Funciona bien en todos los modelos. El enfoque del lenguaje núcleo cae en esta categoría.

# Semántica (4)

## Cuatro enfoques de semántica de LP

- Una semántica axiomática: la semántica de una declaración es la relación entre el estado de entrada (la situación antes de ejecutarse la declaración) y el estado de salida (la situación despues de ejecutarse la declaración). Funciona bien en modelos con estado.
- Una semántica denotacional: una declaración es una función sobre un dominio abstracto. Funciona bien para modelos declarativos, pero es complicado cuando se aplica a lenguajes concurrentes.
- Una semántica lógica: una declaración es un modelo de una teoría lógica. Funciona bien para los modelos de computación declarativa y relacional.
- Una semántica operacional: ejecución de una declaración en términos de una máquina abstracta. Funciona bien en todos los modelos. El enfoque del lenguaje núcleo cae en esta categoría.

# Semántica (4)

## Cuatro enfoques de semántica de LP

- Una semántica axiomática: la semántica de una declaración es la relación entre el estado de entrada (la situación antes de ejecutarse la declaración) y el estado de salida (la situación despues de ejecutarse la declaración). Funciona bien en modelos con estado.
- Una semántica denotacional: una declaración es una función sobre un dominio abstracto. Funciona bien para modelos declarativos, pero es complicado cuando se aplica a lenguajes concurrentes.
- Una semántica lógica: una declaración es un modelo de una teoría lógica. Funciona bien para los modelos de computación declarativa y relacional.
- Una semántica operacional: ejecución de una declaración en términos de una máquina abstracta. Funciona bien en todos los modelos. El enfoque del lenguaje núcleo cae en esta categoría.

# Semántica (4)

## Cuatro enfoques de semántica de LP

- Una semántica axiomática: la semántica de una declaración es la relación entre el estado de entrada (la situación antes de ejecutarse la declaración) y el estado de salida (la situación despues de ejecutarse la declaración). Funciona bien en modelos con estado.
- Una semántica denotacional: una declaración es una función sobre un dominio abstracto. Funciona bien para modelos declarativos, pero es complicado cuando se aplica a lenguajes concurrentes.
- Una semántica lógica: una declaración es un modelo de una teoría lógica. Funciona bien para los modelos de computación declarativa y relacional.
- Una semántica operacional: ejecución de una declaración en términos de una máquina abstracta. Funciona bien en todos los modelos. El enfoque del lenguaje núcleo cae en esta categoría.

# Plan

# Abstracciones lingüísticas

## ¿Qué son?

- Evolución de los LP: p. ej. ciclos `for`. Nueva construcción lingüística.
- Se define en dos fases:
  - Nueva construcción gramatical.
  - Traducción al lenguaje núcleo.
- Ejs.:funciones (`fun`), ciclos (`for`), funciones perezosas (`fun lazy`), clases (`class`), candados (`lock`).

## Abstracción `fun`

- Sintaxis tanto para definir funciones como para invocarlas.

# Abstracciones lingüísticas

## ¿Qué son?

- Evolución de los LP: p. ej. ciclos `for`. Nueva construcción lingüística.
- Se define en dos fases:
  - Nueva construcción gramatical.
  - Traducción al lenguaje núcleo.
- Ejs.:funciones (`fun`), ciclos (`for`), funciones perezosas (`fun lazy`), clases (`class`), candados (`lock`).

## Abstracción `fun`

- Sintaxis tanto para definir funciones como para invocarlas.

Traducción: en términos de definiciones de procedimientos e invocaciones a ellos.

# Abstracciones lingüísticas

## ¿Qué son?

- Evolución de los LP: p. ej. ciclos **`for`**. Nueva construcción lingüística.
- Se define en dos fases:
  - Nueva construcción gramatical.
  - Traducción al lenguaje núcleo.
- Ejs.:funciones (**`fun`**), ciclos (**`for`**), funciones perezosas (**`fun lazy`**), clases (**`class`**), candados (**`lock`**).

## Abstracción `fun`

- Sintaxis tanto para definir funciones como para invocarlas.
- Traducción en términos de definiciones de procedimientos e invocaciones a ellos.

¿Qué significa exactamente `{F1 {F2 X} {F3 Y}}`? (invocación anidada de funciones)? ¿El orden en que se hacen estas invocaciones está definido? Si es así, ¿cuál es ese orden? La traducción clarifica la semántica escogida.

# Abstracciones lingüísticas

## ¿Qué son?

- Evolución de los LP: p. ej. ciclos **`for`**. Nueva construcción lingüística.
- Se define en dos fases:
  - Nueva construcción gramatical.
  - Traducción al lenguaje núcleo.
- Ejs.:funciones (**`fun`**), ciclos (**`for`**), funciones perezosas (**`fun lazy`**), clases (**`class`**), candados (**`lock`**).

## Abstracción **`fun`**

- Sintaxis tanto para definir funciones como para invocarlas.
- Traducción en términos de definiciones de procedimientos e invocaciones a ellos.
- ¿Qué significa exáctamente `{F1 {F2 X} {F3 Y}}` (invocación anidada de funciones)? ¿El orden en que se hacen estas invocaciones está definido? Si es así, ¿cuál es ese orden? La traducción clarifica la semántica escogida.

# Abstracciones lingüísticas

## ¿Qué son?

- Evolución de los LP: p. ej. ciclos **`for`**. Nueva construcción lingüística.
- Se define en dos fases:
  - Nueva construcción gramatical.
  - Traducción al lenguaje núcleo.
- Ejs.:funciones (**`fun`**), ciclos (**`for`**), funciones perezosas (**`fun lazy`**), clases (**`class`**), candados (**`lock`**).

## Abstracción **`fun`**

- Sintaxis tanto para definir funciones como para invocarlas.
- Traducción en términos de definiciones de procedimientos e invocaciones a ellos.
- ¿Qué significa exáctamente `{F1 {F2 X} {F3 Y}}` (invocación anidada de funciones)? ¿El orden en que se hacen estas invocaciones está definido? Si es así, ¿cuál es ese orden? La traducción clarifica la semántica escogida.

# Abstracciones lingüísticas

## ¿Qué son?

- Evolución de los LP: p. ej. ciclos **`for`**. Nueva construcción lingüística.
- Se define en dos fases:
  - Nueva construcción gramatical.
  - Traducción al lenguaje núcleo.
- Ejs.:funciones (**`fun`**), ciclos (**`for`**), funciones perezosas (**`fun`** `lazy`), clases (**`class`**), candados (**`lock`**).

## Abstracción **`fun`**

- Sintaxis tanto para definir funciones como para invocarlas.
- Traducción en términos de definiciones de procedimientos e invocaciones a ellos.
- ¿Qué significa exáctamente `{F1 {F2 X} {F3 Y}}` (invocación anidada de funciones)? ¿El orden en que se hacen estas invocaciones está definido? Si es así, ¿cuál es ese orden? La traducción clarifica la semántica escogida.

# Azúcar sintáctico

## ¿Qué son?

- Notación abreviada para modismos que ocurren con frecuencia.
- Se define análogamente a una abstracción lingüística:
  - Nueva construcción gramatical.
  - Traducción al lenguaje núcleo.
- El azúcar sintáctico no provee una abstracción nueva, solo reduce el tamaño del programa y mejora su legibilidad.

## Ejemplo

En lugar de

```
if N==1 then [1]
else
   local L in
      ...
   end
end
```

podemos escribir

```
if N==1 then [1]
else L in
   ...
end
```

# Azúcar sintáctico

## ¿Qué son?

- Notación abreviada para modismos que ocurren con frecuencia.
- Se define análogamente a una abstracción lingüística:
  - Nueva construcción gramatical.
  - Traducción al lenguaje núcleo.
- El azúcar sintáctico no provee una abstracción nueva, solo reduce el tamaño del programa y mejora su legibilidad.

## Ejemplo

En lugar de

```
if N==1 then [1]
else
   local L in
      ...
   end
end
```

podemos escribir

```
if N==1 then [1]
else L in
   ...
end
```

# Azúcar sintáctico

## ¿Qué son?

- Notación abreviada para modismos que ocurren con frecuencia.
- Se define análogamente a una abstracción lingüística:
  - Nueva construcción gramatical.
  - Traducción al lenguaje núcleo.
- El azúcar sintáctico no provee una abstracción nueva, solo reduce el tamaño del programa y mejora su legibilidad.

## Ejemplo

En lugar de

```
if N==1 then [1]
else
   local L in
      ...
   end
end
```

podemos escribir

```
if N==1 then [1]
else L in
   ...
end
```

# Azúcar sintáctico

## ¿Qué son?

- Notación abreviada para modismos que ocurren con frecuencia.
- Se define análogamente a una abstracción lingüística:
  - Nueva construcción gramatical.
  - Traducción al lenguaje núcleo.
- El azúcar sintáctico no provee una abstracción nueva, solo reduce el tamaño del programa y mejora su legibilidad.

## Ejemplo

En lugar de

```
if N==1 then [1]
else
   local L in
      ...
   end
end
```

podemos escribir

```
if N==1 then [1]
else L in
   ...
end
```

# Desarrollo de este paradigma

# Desarrollo de este paradigma

# Desarrollo de este paradigma

1. El almacén de Variables declarativas
2. Procedimientos y registros como valores
3. El lenguaje núcleo y su semántica
4. La iteración como mecanismo de solución de problemas
5. La recursión como mecanismo de solución de problemas
6. La abstracción procedimental
7. La genericidad
8. La instanciación
9. El embebimiento

# Desarrollo de este paradigma

1. El almacén de Variables declarativas
2. Procedimientos y registros como valores
3. El lenguaje núcleo y su semántica
4. La iteración como mecanismo de solución de problemas
5. La recursión como mecanismo de solución de problemas
6. La abstracción procedimental
7. La genericidad
8. La instanciación
9. El embebimiento

# Desarrollo de este paradigma

1. El almacén de Variables declarativas
2. Procedimientos y registros como valores
3. El lenguaje núcleo y su semántica
4. La iteración como mecanismo de solución de problemas
5. La recursión como mecanismo de solución de problemas
6. La abstracción procedimental
7. La genericidad
8. La instanciación
9. El embebimiento

# Desarrollo de este paradigma

# Desarrollo de este paradigma

# Desarrollo de este paradigma

1. El almacén de Variables declarativas
2. Procedimientos y registros como valores
3. El lenguaje núcleo y su semántica
4. La iteración como mecanismo de solución de problemas
5. La recursión como mecanismo de solución de problemas
6. La abstracción procedimental
7. La genericidad
8. La instanciación
9. El embebimiento

# Desarrollo de este paradigma

1. El almacén de Variables declarativas
2. Procedimientos y registros como valores
3. El lenguaje núcleo y su semántica
4. La iteración como mecanismo de solución de problemas
5. La recursión como mecanismo de solución de problemas
6. La abstracción procedimental
7. La genericidad
8. La instanciación
9. El embebimiento