Paradigmas de lenguajes de programación

Alejandro Ríos

Departamento de Computación, FCEyN, UBA

"A language that doesn't affect the way you think about programming, is not worth knowing"

Epigrams in Programming, Alan Perlis (primer ganador del Turing Award), ACM SIGPLAN, Sept. 1982

15 de agosto de 2017

Cátedra y modalidad

- Modalidad: Clases teóricas y prácticas

Teoría

- Alejandro Ríos (`rios@dc.uba.ar`)
- (típicamente) Jueves: 17:00hs a 19:30hs
- **Invitados:**
  Fidel(=Pablo E. Martínez López) a una clase de PF y una clase de mónadas.
  Hernán Wilkinson a una clase de Metaprogramación.

Práctica

- Christian Roldán (JTP)
  Gabriela Steren, Edgardo Zoppi, Julián Dabbah, Daniel Álvarez, Sabrina Izcovich (Ays. 1a)
  Sebastián Vita, Iván Arcuschin (Ays. 2a)
- (típicamente) Martes: 17:30hs a 21:00hs

Recursos

Bibliografía

- Textos: no hay un texto principal, se utilizan varios, referencias en página web
- Apuntes: serán introducidos oportunamente
- Publicaciones relacionadas
- Diapositivas de teóricas y prácticas
- Guías de ejercicios

Página web

- Información al día del curso, consultar periódicamente y leer al menos una vez todas las secciones

Mailing list

- ¡Hacer todas las preguntas y consultas que quieran!

Recursos

Software

- Haskell
  - El intérprete (Hugs): `http://www.haskell.org`
  - Documentos (accesibles en el mismo sitio):
    - *A Gentle Introduction to Haskell*, Paul Hudak, John Peterson y Joseph H. Fasel.
    - *The Hugs 98 User Manual*, Mark P. Jones y John Peterson, 1999. (Manual del usuario; modestas 84 páginas)
    - *Haskell 98 Language and Libraries - The Revised Report*, Simon Peyton Jones (ed.), 2002. (La referencia definitiva sobre Hugs98, 277 páginas....)
- SWI-Prolog (programación lógica)
- Pharo (Smalltalk - programación orientada a objetos)

Paradigma

Marco filosófico y teórico de una escuela científica o disciplina en la que se formulan teorías, leyes y generalizaciones y se llevan a cabo experimentos que les dan sustento

Fuente: Merriam-Webster[1]

[1] *A philosophical and theoretical framework of a scientific school or discipline within which theories, laws, and generalizations and the experiments performed in support of them are formulated*

Lenguajes de Programación

- lenguaje usado para comunicar instrucciones a una computadora
- las instrucciones describen cómputos que llevará a cabo la computadora
- la noción de cómputo puede formalizarse de muchas maneras:
  - máquinas de Turing
  - cálculo lambda
  - funciones recursivas
  - ...

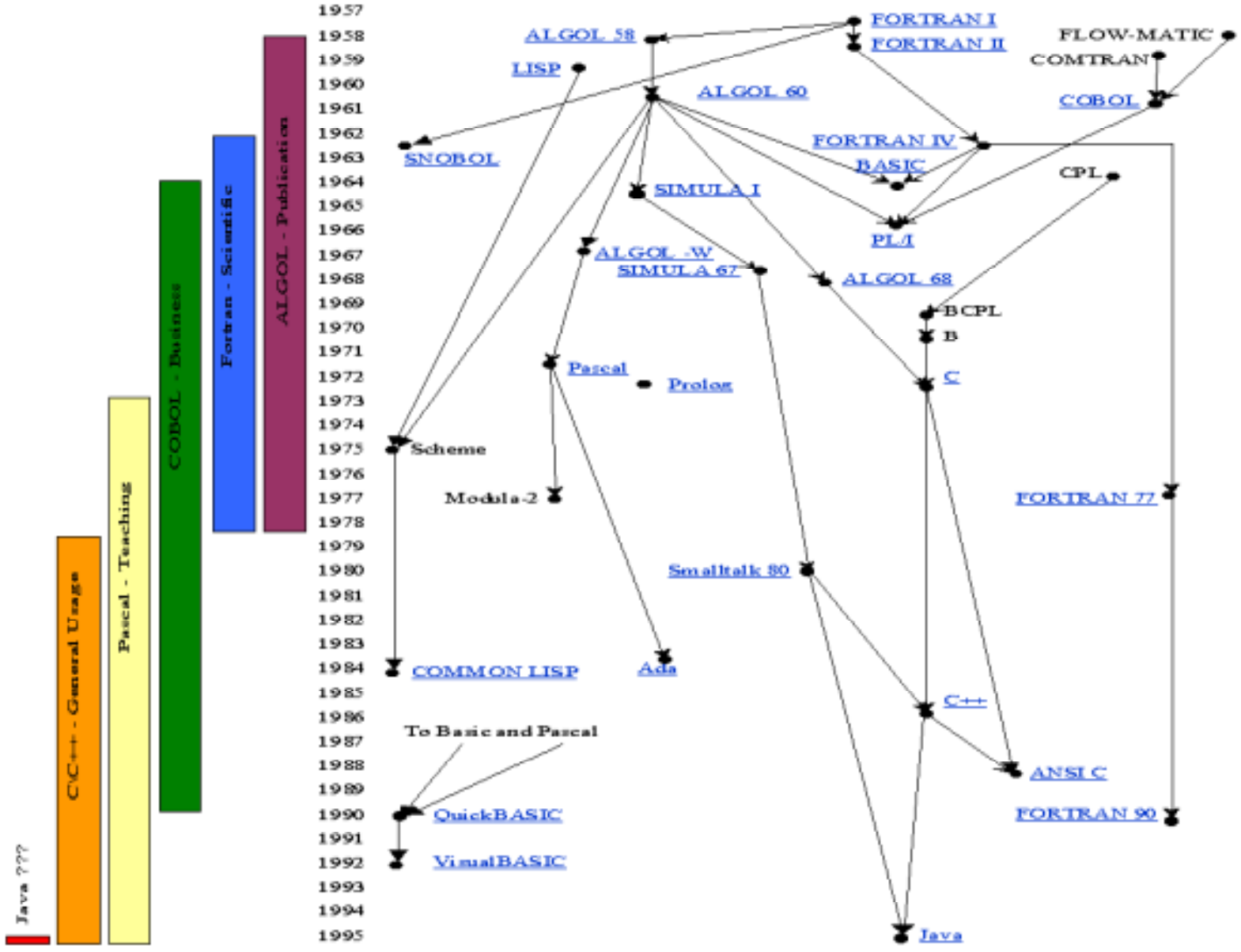
  Siempre se obtiene la misma clase de funciones computables!!
- el lenguaje es computacionalmente completo si puede expresar todas las funciones computables

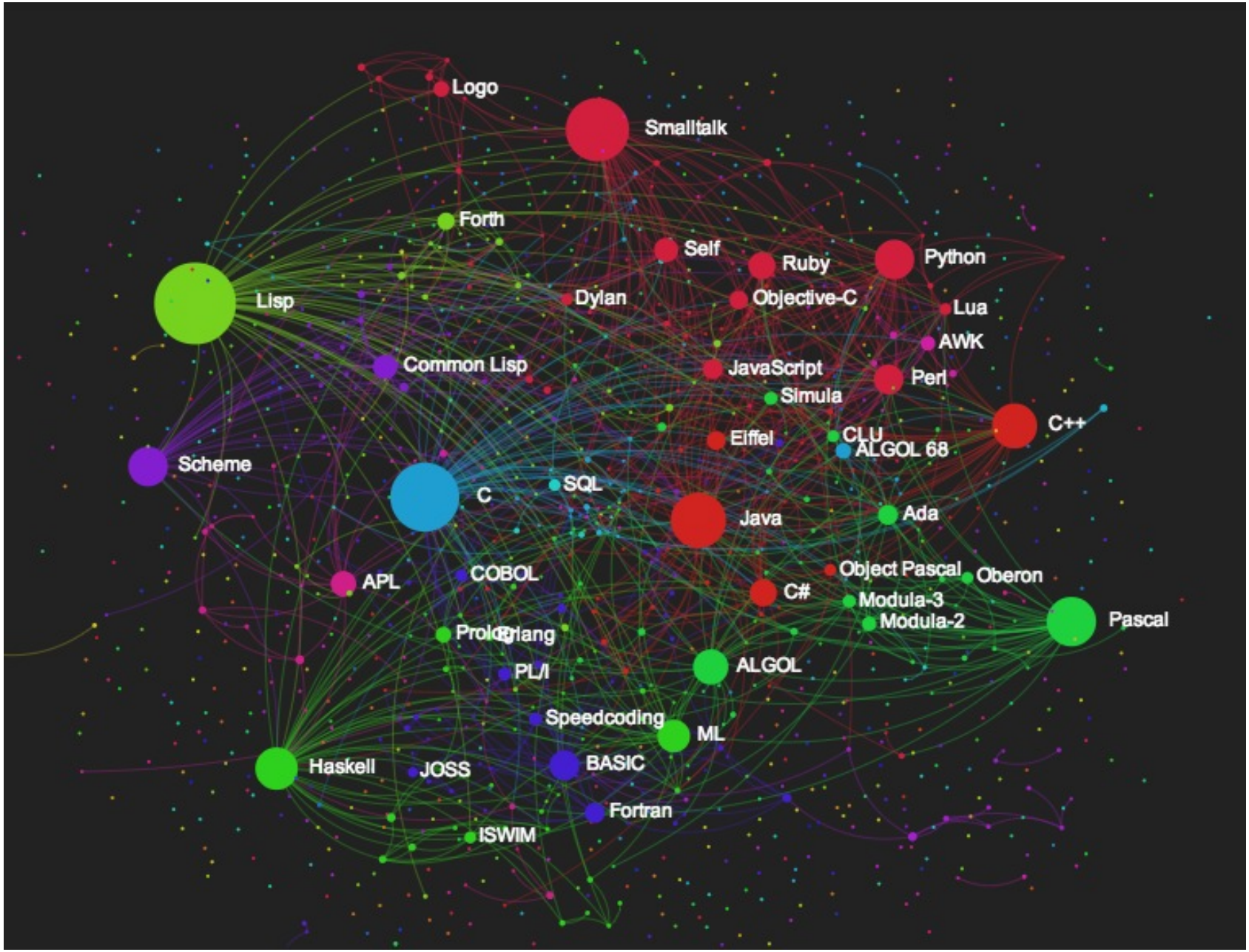
Paradigmas de lenguajes de Programación

Marco filosófico y teórico en el que se formulan soluciones a problemas de naturaleza algorítmica

- Lo entendemos como
  - un estilo de programación
  - en el que se escriben soluciones a problemas en términos de algoritmos
- El ingrediente básico es el modelo de cómputo
  - la visión que tiene el usuario de cómo se ejecutan los programas

Orígenes de los Lenguajes de Programación





exploringdata.github.io/info/programming-languages-influence-network/

Addendum

Griff: July 17, 2012 at 10:01 am How can something influence everything but be used by no one? Haskell's life force is in the things it influenced surely [snip]

Aphidman: July 19, 2012 at 2:25 pm Maybe Haskell is the Velvet Underground of programming languages. They say that the Velvet Underground only sold 10,000 records, but everyone who bought one went out and started a rock band.

`http://griffsgraphs.com/2012/07/01/programming-languages-influences/`

Objetivos del curso

Conocer los pilares conceptuales sobre los cuales se erigen los lenguajes de programación de modo de poder

- Comparar lenguajes
- Seleccionar el más adecuado para una determinada tarea
- Usar las herramientas disponibles adecuadamente
- Prepararse para lenguajes/paradigmas futuros

Enfoque del curso

1. Angulo conceptual
   - Introducción informal de conceptos a través de ejemplos.
2. Angulo de fundamentos
   - Introducir las bases rigurosas (lógicas y matemáticas) sobre las que se sustentan cada uno de los paradigmas o parte de los mismos

Algunos temas cubiertos - Angulo conceptual

- Recursión
- Valores, expresiones, currificación, funciones de alto orden, polimorfismo paramétrico, esquemas de recursión
- Asignación y efectos laterales
- Expresiones de tipo, sistema de tipos, checkeo de tipos, inferencia de tipos
- Resolución en lógica proposicional y de primer orden, Cláusulas de Horn, unificación, refutación, resolución SLD
- Objeto, clase, herencia, method dispatch estático/dinámico, polimorfismo de subclase, subtipos, sistemas de tipos invariantes

Algunos temas cubiertos - Angulo de fundamentos

- Lambda cálculo
  - Sintaxis y ejemplos de programación
  - Sistemas de tipos e inferencia
  - Semántica operacional
- Resolución
  - En lógica proposicional
  - En lógica de primer orden
  - SLD
- Programación orientada a objetos
  - Sistemas de tipos, subtipado.

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica

Definición de un Lenguaje

- Sintaxis
  - descripción del conjunto de secuencias de símbolos considerados como programas válidos
  - teoría de lenguajes formales bien desarrollada, comenzando a mediados de 1950 con Noam Chomsky como pionero
  - notación BNF (y EBNF) ampliamente utilizada; desarrollada por John Backus para Algol 58, modificada por Peter Naur para Algol 60
  - amplio abanico de herramientas para generar analizadores léxicos y parsers a partir de notación formal como BNF
- Sistema de Tipos
- Semántica

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
  - propósito: prevenir errores en tiempo de ejecución
  - en general, requiere anotaciones de tipo en el código fuente
  - ejemplos: evitar sumar booleanos, aplicar función a número incorrecto de argumentos.
  - análisis de tipos en tiempo de compilación: chequeo de tipos estático
  - análisis de tipos en tiempo de ejecución: chequeo de tipos dinámico
  - veremos en detalle en el Eje de Fundamentos
- Semántica

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica
  - descripción del significado de instrucciones y expresiones
  - puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
  - ¿Por qué semántica formal?[a]:
    - Destructor británico H.M.S. Sheffield hundido en guerra de Malvinas. El radar de alerta de la nave estaba programado para identificar el misil Exocet como .aliado"debido a que el arsenal Inglés incluye el "homing device" de los Exocet y permitió que el misil alcanzara su blanco (el H.M.S. Sheffield).

[a] `http://www.cs.tau.ac.il/~nachumd/horror.html`

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica
  - descripción del significado de instrucciones y expresiones
  - puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
  - ¿Por qué semántica formal?
    - Votos perdidos por computadora en Toronto. El distrito de Toronto finalmente abandonó votación electrónica.

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica
  - descripción del significado de instrucciones y expresiones
  - puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
  - ¿Por qué semántica formal?
    - 225 de los 254 pasajeros de Korean Airlines KAL 901 en Guam fallecen en accidente. Bug descubierto en altímetro barométrico del Ground Proximity Warning System (GPWS).

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica
  - descripción del significado de instrucciones y expresiones
  - puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
  - ¿Por qué semántica formal?
    - Falla en el despegue del satélite Ariane 5 causado por software error en la rutina de manejo de excepciones resultante de una mala conversión de un punto flotante de 64-bit a un entero.

Paradigmas de Lenguajes

- Nos ocuparemos de los paradigmas:
  - *imperativo*
  - funcional
  - lógico
  - orientado a objetos
- Hay otros: concurrente, eventos, continuaciones, quantum, etc.
- ¡La distinción a veces no está clara!

Imperativo

¿Qué hace este programa?

```
int main (){
  int r,s,n;
  n = 1;
  r = 1;
  s = 0;
  while (n<11) {
    r = r * n;
    s = s + r;
    n = n + 1;
  };
  printf ("%d",s);
}
```

Ayuda: imprime 4037913

Imperativo

¿Qué hace este programa?

```
int main (){
  int r,s,n;
  n = 1;
  r = 1;
  s = 0;
  while (n<11) {
    r = r * n;
    s = s + r;
    n = n + 1;
  };
  printf ("%d",s);
}
```

Ayuda: imprime 4037913

Estado global + asignación + control de flujo

Evaluación

- Computación expresada a través de modificación reiterada de estado global (o memoria)
- Variables como abstracción de celdas de memoria
- Resultados intermedios se almacenan en la memoria
- Repetición basada en iteración

Imperativo

¿Qué hace este programa?

```
int main (){
  int r,s,n;
  n = 1;
  r = 1;
  s = 0;
  while (n<11) {
    r = r * n;
    s = s + r;
    n = n + 1;
  };
  printf ("%d",s);
}
```

Ayuda: imprime 4037913

Evaluación

- Eficiente
  - Modelo ejecución - Arquit. $= 0$
- Bajo nivel de abstracción
  - Especif - Implement $= \infty$
- Matemática/lógica de programas compleja
  - Operacional vs denotacional

Imperativo - Ejemplo - Bajo nivel de abstracción

¿Qué hace este programa?

```
int g;
int f(int x)
   ++g;
   return 3;

int main ()
  g=6;
  if (g==f(2)+f(2))
    printf("Eject!");
  else
    printf("Mantener
      curso actual!");
```

Imperativo - Ejemplo - Bajo nivel de abstracción

¿Qué hace este programa?

```
int g;
int f(int x)
   ++g;
   return 3;

int main ()
  g=6;
  if (g==f(2)+f(2))
    printf("Eject!");
  else
    printf("Mantener
      curso actual!");
```

- El resultado depende del orden de evaluación de $==$.
- No habría que preocuparse por dicho orden

Funcional

```
sum (map fact [1..10])

fact 0 = 1
fact n = n * fact (n-1)
```

Funcional

```
sum (map fact [1..10])

fact 0 = 1
fact n = n * fact (n-1)
```

Evaluación

- Computación expresada a través de la aplicación y composición de funciones
- No hay un estado global
- Resultados intermedios (salida de las funciones) son pasados directamente a otras funciones como argumentos
- Repetición basada en recursión
- Expresiones tipadas

Funcional

```
sum (map fact [1..10])

fact 0 = 1
fact n = n * fact (n-1)
```

Evaluación

- Alto nivel de abstracción
  - Especificación ejecutable: *Sumar resultado de aplicar factorial a la lista de números de 1 a n*
- Declarativo
- Matemática de programas elegante
- Razonamiento algebraico posible
  - $e + e \simeq 2 * e$
  - $a == b \simeq b == a$
- Ejecución lenta (?)

Lógico

```
fact(0,1).
fact(N,Res) :- M is N-1, fact(M,Aux), Res is N*Aux.
lFact([],N,N).
lFact([H|T],M,N) :- fact(M,H), M1 is M+1, lFact(T,M1,N).
sumaFact(X,N) :- lFact(L,1,N), sumList(L,X).
```

Evaluación

- Programas son predicados
- Computación expresada a través de "proof search"
- No hay estado global
- Resultados intermedios son pasados a través de unificación
- Repetición basada en recursión

Lógico

```
fact(0,1).
fact(N,Res) :- M is N-1, fact(M,Aux), Res is N*Aux.
lFact([],N,N).
lFact([H|T],M,N) :- fact(M,H), M1 is M+1, lFact(T,M1,N).
sumaFact(X,N) :- lFact(L,1,N), sumList(L,X).
```

Evaluación

- Alto nivel de abstracción
  - Especificación ejecutable
- Declarativo
- Fundamento lógico robusto
  - Resolución
- Ejecución lenta

Orientado a Objetos

```
Numero << factorial

self = 0 ifTrue: [^ 1].
self > 0 ifTrue: [^ self * (self - 1) factorial].
self error: 'Not valid for negative integers'

Numero << sumaFactoriales

^((1 to: self) collect: [:x | x factorial]) sum
```

Evaluación

- Computación a través del intercambio de mensajes entre objetos
- Objetos se agrupan en clases, clases se agrupan en jerarquías

Orientado a Objetos

Evaluación

- Alto nivel de abstracción
  - Objetos
  - Clases
  - Mensajes
- Arquitectura extensible
  - Jerarquía de clases
  - Polimorfismo de subtipos
  - Binding dinámico
- Matemática de programas compleja

Top five: "Great Works in Programming Languages", B.Pierce

- C.A.R. Hoare. An axiomatic basis for computer programming. Communications of the ACM, 12(10):576-580 and 583, October 1969.
- Peter J. Landin. The next 700 programming languages. Communications of the ACM, 9(3):157-166, March 1966.
- Robin Milner. A theory of type polymorphism in programming. Journal of Computer and System Sciences, 17:348-375, August 1978.
- Gordon Plotkin. Call-by-name, call-by-value, and the λ-calculus. Theoretical Computer Science, 1:125-159, 1975.
- John C. Reynolds. Towards a theory of type structure. In Colloque sur la Programmation, Paris, France, volume 19 of Lecture Notes in Computer Science, pages 408-425. Springer-Verlag, 1974.

Vanguardia

Conferencias Europeas

- The European Joint Conferences on Theory and Practice of Software (ETAPS)
  - Foundations of Software Science and Computation Structures (FOSSACS)
  - Fundamental Approaches to Software Engineering (FASE)
  - European Symposium on Programming (ESOP)
  - International Conference on Compiler Construction (CC)
  - Tools and Algorithms for the Construction and Analysis of Systems (TACAS)
- International Colloquium on Automata, Languages and Programming (ICALP)
- Computer Science Logic

Vanguardia

Conferencias ACM

- Principles of Programming Languages (POPL)
- International Conference on Functional Programming (ICFP)
- Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)
- Programming Language Design and Implementation (PLDI)
- Principles and Practice of Parallel Programming (PPOPP)