

# historia de los lenguajes de programación

Paradigmas de la Programación  
FaMAF 2021

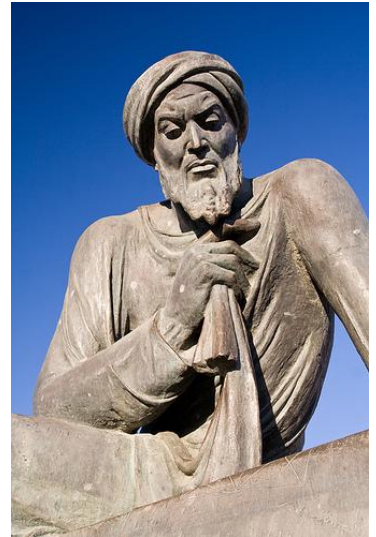
basado en filminas de Vitaly Shmatikov

[http://www.cs.utexas.edu/~shmat/courses/cs345/03  
history.ppt](http://www.cs.utexas.edu/~shmat/courses/cs345/03history.ppt)

precursores

# Algoritmo

- Abu Ja'far Muhammad ibn Musa **al-Khorezmi** ("originario de Khorezm")
  - vivió en Bagdad en 780 – 850 aC
  - matemático del Califa
  - autor de "Breve introducción al cálculo usando reglas de completitud y reducción"
  - eliminar unidades negativas de la ecuación añadiendo la misma cantidad en el otro miembro



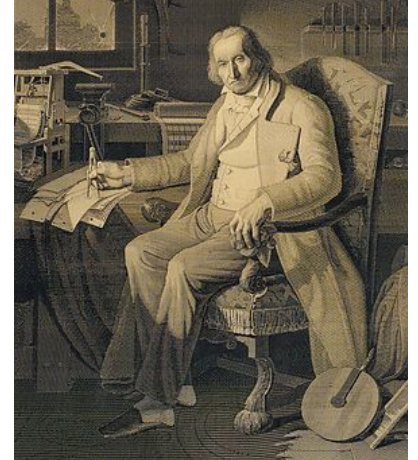
# “Cálculo del pensamiento”

- Gottfried Wilhelm Leibniz
  - 1646 - 1716
  - Inventor del cálculo y del sistema binario
  - “Calculus ratiocinator”: el razonamiento se puede reducir a un lenguaje formal simbólico, y todos los argumentos se pueden resolver mediante la manipulación mecánica de conceptos lógicos
  - inventó la calculadora mecánica



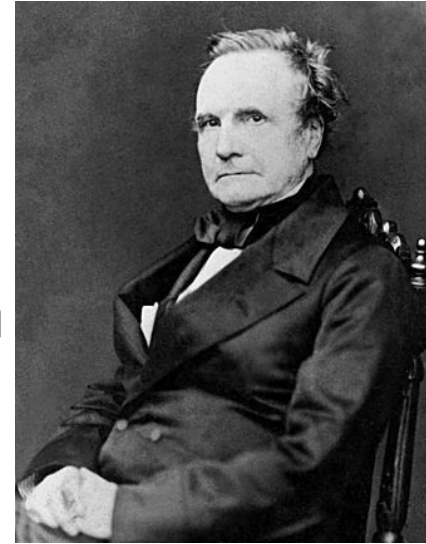
# el telar de Jacquard

- demo en 1801
- se programaban los diseños en el telar mediante tarjetas perforadas
- totalmente pegado al hardware: cada hueco se correspondía con una acción del telar
- no estaba hardcodeado



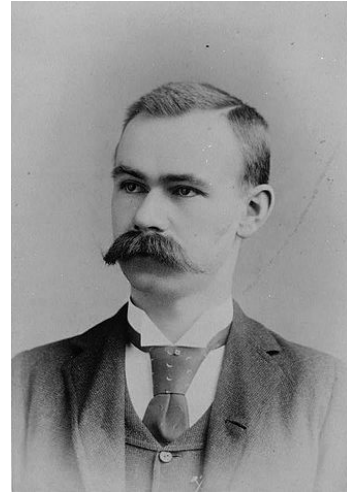
# Analytical Engine

- Charles Babbage (1791 – 1871) inventa (pero nunca llega a construir) la primera computadora (no sólo calculadora) programable
- podría hacer bucles y condicionales, tenía memoria integrada
- la primera máquina Turing-completa?
- Ada Lovelace (1815 – 1852) implementó el primer algoritmo para el Analytical Engine, para calcular una secuencia de números de Bernoulli

A detailed image of Ada Lovelace's algorithm for the Analytical Engine, showing a complex table of operations and calculations. The table is organized into columns for 'Operation', 'Data', and 'Result'. It includes various mathematical expressions and logical instructions, such as 'If the result of the operation is greater than zero, then go to the operation numbered 1000'. The table is a handwritten manuscript on aged paper, with some parts crossed out and others corrected.

# el tabulador estadístico

- Herman Hollerith (1860-1929)  
desarrolla un tabulador para obtener estadísticas de grandes cantidades de datos
- Funda la Tabulating Machine Company, que termina fusionándose para formar IBM
- es el padre del procesamiento automático de datos



Formalismos para computación



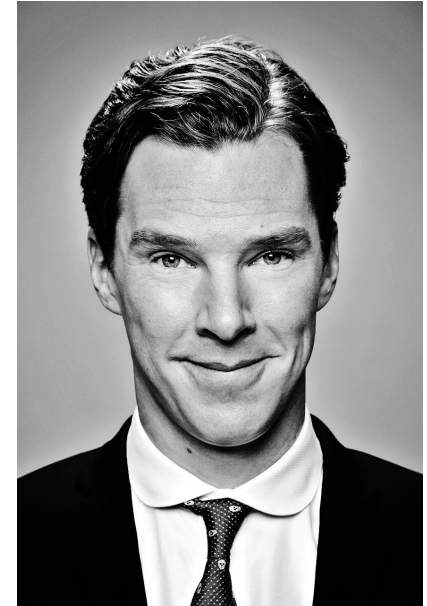
- Lógica de predicados
  - Gottlob Frege (1848-1925)
  - base formal para la teoría de la demostración y la prueba de teoremas automatizada
  - la computación como deducción

**programación lógica**



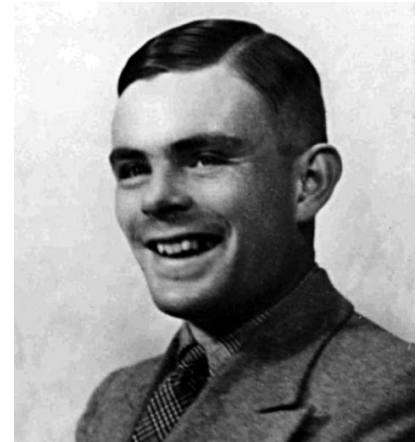
- máquinas de Turing
  - Alan Turing (1912-1954)
  - secuencias de comandos, transiciones entre estados explícitas, actualización mediante asignación

**programación imperativa**



- máquinas de Turing
  - Alan Turing (1912-1954)
  - secuencias de comandos, transiciones entre estados explícitas, actualización mediante asignación

**programación imperativa**

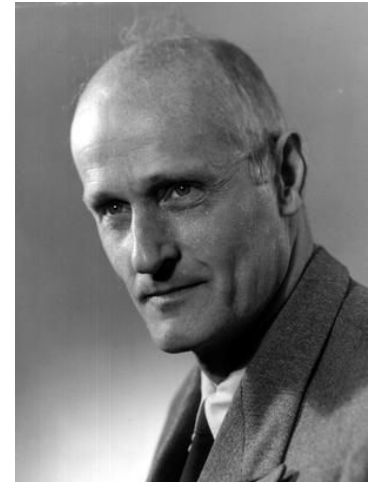


- Lambda cálculo
  - Alonzo Church (1903-1995)
  - bases formal para todos los lenguajes funcionales, semántica, teoría de tipos
  - evaluación de expresiones puras, sin asignación

**programación funcional**



- funciones recursivas y autómatas
  - Stephen Kleene (1909-1994)



# Tesis de Church-Turing

- todos los formalismos sintácticos describen la misma clase de objetos matemáticos
  - tesis de Church: “Toda función calculable (predicado calculable) es general recursivo”
  - tesis de Turing: “toda función que se consideraría naturalmente computable es computable por una máquina de Turing”
- el lambda-cálculo y las máquinas de Turing tienen un poder expresivo equivalente

Lenguajes de programación

# qué es un lenguaje de programación?

- una notación formal para especificar computaciones
  - sintaxis (definida por una gramática independiente de contexto)
  - semántica para cada construcción sintáctica
  - implementación práctica en una máquina (real o virtual, hay que encontrar el balance entre portabilidad y eficiencia)



# lenguajes ensamblador

- la evolución de las tarjetas perforadas
- los inventan los diseñadores de hardware a principio de los 1950
- usan nombres que se lugar de binarios

**push ebp**  
**mov ebp, esp**  
**sub esp, 4**  
**push edi**



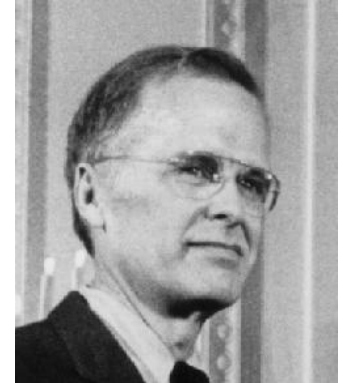
- tienen macros reusables y subrutinas

# COBOL

- diseñado 1959, estándar en 1968
- Grace Hopper y el Departamento de Defensa de los Estados Unidos
- trata de imitar el lenguaje natural, para facilitar su uso por no computólogos y ser su propia documentación
- tremendamente verboso
- muy poco estructurado
- muy utilizado históricamente, y hoy en sistemas legacy



# FORTRAN

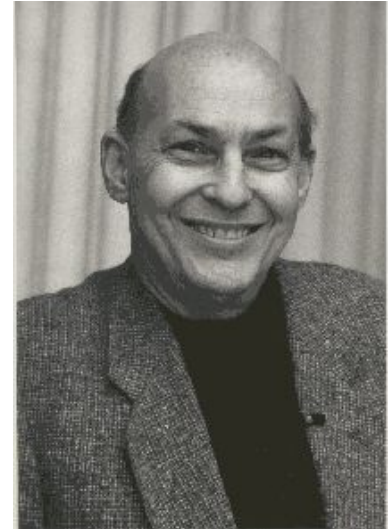


- lenguaje procedural imperativo
    - se usa en algunos cálculos científicos
  - desarrollado en IBM en los 1950s por John Backus (1924-2007)
    - Backus aboga por la programación funcional en su charla del premio Turing 1977
- "We did not know what we wanted and how to do it. It just sort of grew. The first struggle was over what the lenguaje would look like. Then **how to parse expressions** – it was a big problem..."*
- BNF: forma de Backus-Naur para definir gramáticas independientes de contexto

# de FORTRAN a LISP

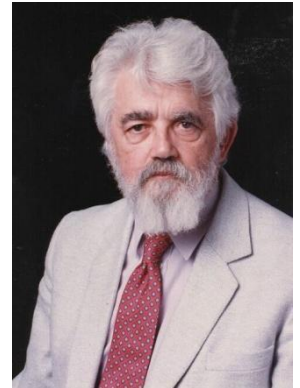
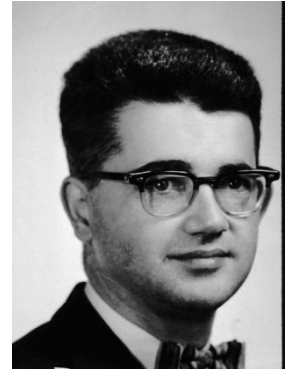
*“Anyone could learn Lisp in one day,  
except that if they already knew FORTRAN,  
it would take three days”*

- Marvin Minsky



# LISP

- Inventado por John McCarthy
- notación formal para lambda-cálculo
- la primera instanciación de muchos conceptos de lenguajes de programación:
  - manejo de memoria automatizado (garbage collection)
  - tipado dinámico
  - no se distingue entre el código y los datos
- todavía se usa: ACL2, Scheme, ...



# LISP Quotes



- “The greatest single programming language ever designed” --Alan Kay
- “LISP being the most powerful and cleanest of languages, that's the language that the GNU project always prefer” -- Richard Stallman
- “Programming in Lisp is like playing with the primordial forces of the universe. It feels like lightning between your fingertips.” -- Glenn Ehrlich
- “Lisp has all the visual appeal of oatmeal with fingernail clippings mixed in” -- Larry Wall
- “LISP programmers know the value of everything and the cost of nothing” -- Alan Perlis


# Algol 60



- Diseñado en 1958-1960
- Gran influencia en lenguajes modernos
  - especifican la sintaxis formalmente con BNF
  - alcance léxico: begin ... end or {...}
  - procedimientos modulares, recursivos, declaración del tipo de las variables
- “Birth of computer science” -- Dijkstra
- “A lenguaje so far ahead of its time that it was not only an improvement on its predecessors, but also on nearly all its successors” -- Hoare

# ejemplo de Algol 60

```
real procedure average(A,n);  
  real array A; integer n;  arreglo sin límites  
begin  
  real sum; sum := 0;  
  for i = 1 step 1 until n do  
    sum := sum + A[i];  
  average := sum/n  sin ; acá  
end;
```

 el retorno del procedimiento es mediante asignación de valor a variable



# una rareza de Algol

- pregunta
  - $x := x$  es equivalente a hacer nada (**skip**)?
- respuesta interesante en Algol

```
integer procedure p;  
begin
```

```
  ...
```

```
    p := p
```

```
  ...
```

```
end;
```

- la asignación acá es en realidad una llamada recursiva

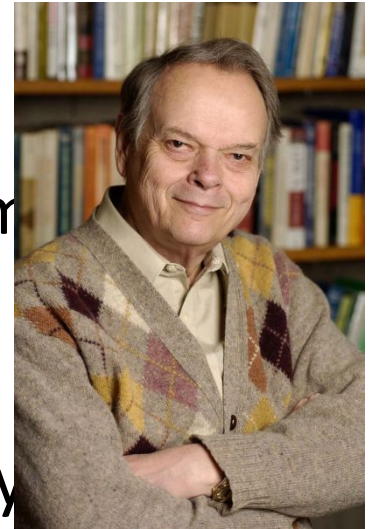
# algunos problemas de Algol 60

- poca disciplina de tipos
  - los parámetros pueden ser arreglos, y los arreglos no tienen límites
  - los parámetros pueden ser procedimientos, pero los tipos de los procedimientos no están especificados
- métodos para pasaje de parámetros
  - Pass-by-name tenía varias anomalías
    - “regla de copia” basada en sustitución, tiene efectos colaterales
  - Pass-by-value es costoso para arreglos
- cuestiones extrañas en control
  - un “goto” fuera de un bloque requiere manejo de memoria

# Algol 60 Legacy

“Another line of development stemming from Algol 60 has led to languages such as Pascal and its descendants, e.g., Euclid, Mesa, and Ada, which are significantly lowerlevel than Algol. Each of these languages seriously restricts the block or procedure mechanism of Algol by eliminating features such as call by name, dynamic arrays, or procedure parameters.”

- John C. Reynolds



# Algol 68

- un sistema de tipos muy elaborado
  - conversiones de tipo complicadas
  - terminología complicada
    - los tipos se llaman “modos”
    - los arreglos se llaman “valores múltiples”
- gramáticas de van Wijngaarden en lugar de BNF (sensibles al contexto)
- no usa pass-by-name
- considerado difícil de entender

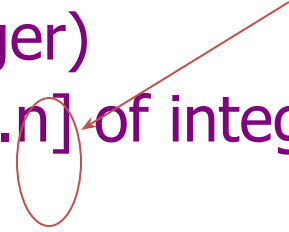


# Pascal

- Niklaus Wirth
- revisa el sistema de tipos de Algol
  - buenos conceptos de estructuras de datos
    - Records, variantes, subrangos
  - más restrictivo que Algol 60/68
    - los parámetros de procedimiento no pueden tener parámetros de procedimiento
- lenguaje popular para enseñanza



# Limitaciones de Pascal

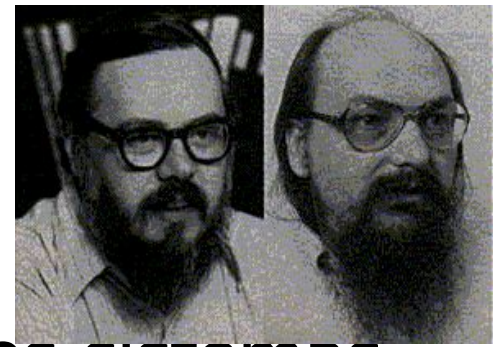
- los límites de un arreglo son parte del tipo illegal  
    procedure p(a: array [1..10] of integer)  
    procedure p(n: integer, a: array [1..n] of integer)  
    
- dificulta el reuso del código:
  - el parámetro tiene que tener un tipo
  - los tipos no pueden tener variables
- no es útil para proyectos industriales, el foco está en la enseñanza

# SIMULA 67

- Ole-Johan Dahl (1931-2002)
- Kristen Nygaard (1926-2002)
- el primer lenguaje orientado a objetos (1962)
  - Objetos y clases
  - subclases y herencia
  - polimorfismo
  - procedimientos virtuales



# BCPL / B / C



- nacido de la frustración con grandes sistemas operativos y grandes lenguajes (Multics, PL/I, Algol 68)
- mantiene alcance léxico y recursión
- acceso al bajo nivel de la máquina
  - manejo de memoria manual
  - manejo de punteros explícito
  - tipado débil
- programación de sistemas operativos para máquinas con poca memoria
  - un “lenguaje ensamblador portable”



# BCPL



- Martin Richards (1966)
- portabilidad y facilidad de compilación

*"The philosophy of BCPL is not one of the tyrant who thinks he knows best and lays down the law on what is and what is not allowed; rather, BCPL acts more as a servant offering his services to the best of his ability without complaint, even when confronted with apparent nonsense. The programmer is always assumed to know what he is doing and is not hemmed in by petty restrictions."*

- un solo tipo (word), equivalencia de punteros y arreglos, aritmética de punteros

# Arreglos y punteros

- un arreglo es un puntero a su primer elemento
- BCPL: `let V = vec 10`  
`V[i]` para indexar el  $i^{\text{ésimo}}$  elemento
- C: `A[i]` es equivalente a la dereferencia de puntero `*((A) + (i))`

# B



- Ken Thompson
- Sintaxis compacta
  - compilador de un solo paso, con poca memoria
  - sin alcances anidados
  - asignación: **=** en lugar del asimétrico **:=**
  - notación pre-/postfija: **x++** en lugar de **x:=x+1**

# divirtiéndonos con prefijo y postfijo

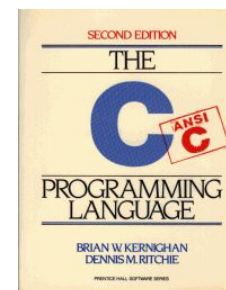
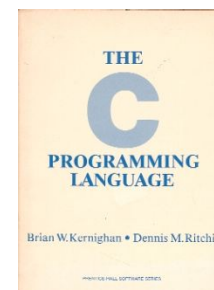
qué significan estos?

$x += x++$

$++x + x++$

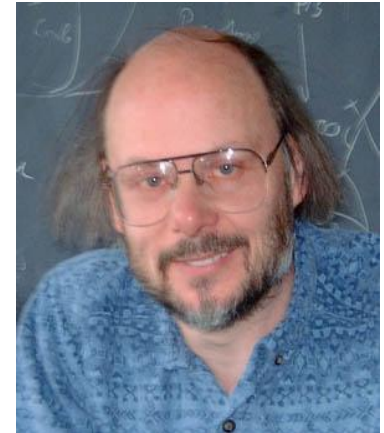
# C

- Bell Labs 1972 (Dennis Ritchie)
- desarrollo relacionado con UNIX
- añade tipado débil a B
  - int, char, y sus tipos punteros
- Compila a código nativo



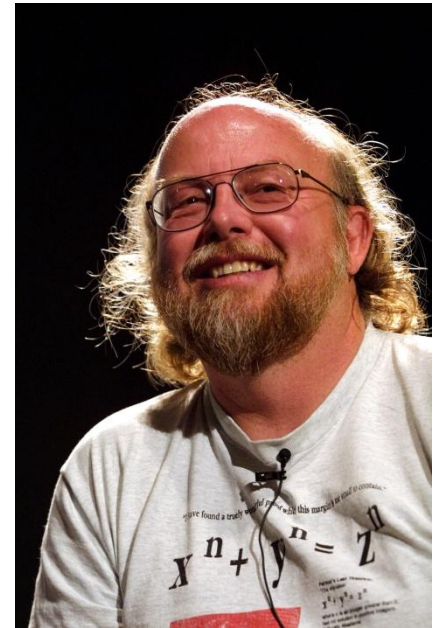
# C++

- Bell Labs 1979 (Bjarne Stroustrup)
  - “C con clases” (C++ desde 1983)
- influenciado por Simula
- originalmente se traducía a C con Cfront, después con compiladores nativos (GNU g++)
- incorpora:
  - herencia múltiple
  - Templates / genéricos
  - manejo de excepciones



# Java

- Sun 1991-1995 (James Gosling)
- mezcla de C y Modula-3
  - a diferencia de C++
    - no usa templates (más adelante provee genéricos), no implementa herencia múltiple, no permite sobrecarga de operadores
  - como Modula-3
    - interfaces explícitas, herencia múltiple, manejo de excepciones, modelo de threading incorporado, referencias y garbage collection automático (sin punteros!!)



# otros lenguajes importantes

- tipo Algol
  - Modula, Oberon, Ada
- Funcional
  - ISWIM, FP, SASL, Miranda, Haskell, LCF, ML, Caml, Ocaml, Scheme, Common LISP
- orientado a objetos
  - Smalltalk, Objective-C, Eiffel, Modula-3, Self, C#, CLOS
- programación lógica
  - Prolog, Gödel, LDL, ACL2, Isabelle, HOL





# ... y más

- procesamiento de datos y bases de datos
  - Cobol, SQL, 4GLs, XQuery
- programación de sistemas
  - PL/I, PL/M, BLISS
- aplicaciones específicas
  - APL, Forth, Icon, Logo, SNOBOL4, GPSS, Visual Basic
- Concurrentes, paralelos, distribuidos
  - Concurrent Pascal, Concurrent C, C\*, SR, Occam, Erlang, Obliq

# scripting

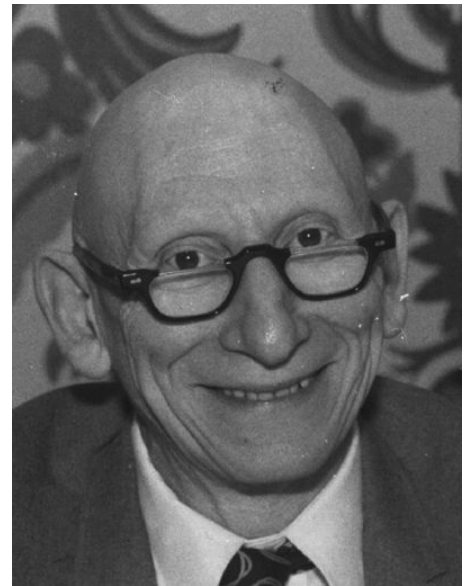
- “mini-lenguajes”
  - awk, make, lex, yacc, autoconf ...
- Command shells, scripting y lenguajes “web”
  - sh, csh, tcsh, ksh, zsh, bash ...
  - Perl, JavaScript, PHP, Python, Rexx, Ruby, Tcl, AppleScript, VBScript ...

*“PHP is a minor evil perpetrated by incompetent amateurs, whereas Perl is a great and insidious evil, perpetrated by skilled but perverted professionals.” - Jon Ribbens*
- frameworks y tecnologías para aplicaciones web
  - ASP.NET, AJAX, Flash, Silverlight ...
    - **Nota:** HTML/XML son lenguajes de marcado, pero a veces contienen scripts ejecutables como Active Server Pages (ASPs) & Java Server Pages (JSPs)

# por qué tantos lenguajes?

“There will always be things we wish to say in our programs that in all lenguajes can only be said poorly.”

- Alan Perlis



# qué motiva la evolución de los lenguajes?

- cómo construir mejores herramientas de software para resolver problemas computacionales
- algunos conceptos útiles evolucionan en diseño de lenguajes
  - Algol → Simula → Smalltalk → C with Classes → C++
- la necesidad de hacer las cosas rápido
  - lenguajes de scripting: Perl, Tcl, Python, PHP, etc.

# qué tienen en común?

- estructura y análisis léxico
  - tokens: keywords, operadores, símbolos, variables
  - expresiones regulares y autómatas
- estructura y análisis sintácticos
  - parsing, gramáticas independientes de contexto
- cosas prácticas
  - alcance, estructura de bloques, variables locales
  - procedimientos, pasaje de parámetros, iteración, recursión
  - chequeo de tipos, estructuras de datos
- semántica
  - qué significan los programas y su correctitud

# historias alternativas

- <http://james-iry.blogspot.com.ar/2009/05/brief-incomplete-and-mostly-wrong.html>
- y una un poco más seria:
- <http://www.cse.buffalo.edu/~shapiro/Courses/CSE305/Notes/notes2.html>

y si se aburren...

- <http://c2.com/cgi/wiki>