

Fundamentos de lenguajes de programación

Introducción al curso

carlos.andres.delgado@correounalvalle.edu.co

Facultad de Ingeniería. Universidad del Valle

Junio de 2020



Contenido

1 Conceptos

2 Un poco de historia

3 Perspectiva de los paradigmas de programación

4 Motivación del curso

Contenido

1 Conceptos

2 Un poco de historia

3 Perspectiva de los paradigmas de programación

4 Motivación del curso

Lenguaje

- Lenguaje \Rightarrow Comunicación
- Lenguaje de programación \Rightarrow Comunicación con la máquina
- Lenguaje hablado y lenguaje escrito
- Lenguaje escrito \Rightarrow Formalismos \Rightarrow Lenguajes formales

Sistemas de escritura

- Sistemas de escrituras independientes del medio:
 - Sumerios
 - Egipcios
 - Mayas



Figura: Ejemplo de escritura egipcia, tomado de enciclopedia britannica

Sistemas de escritura

Evolución de los alfabetos: Pictogramas a Letras.

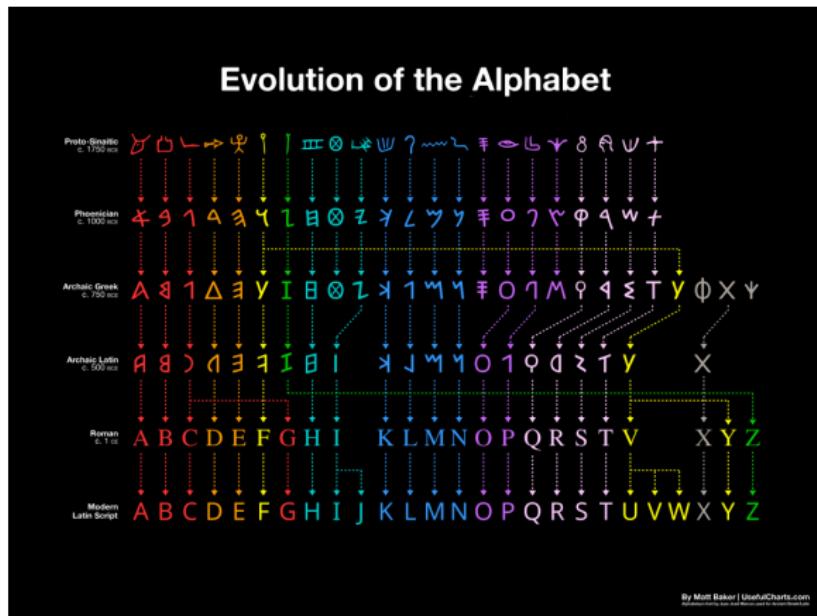
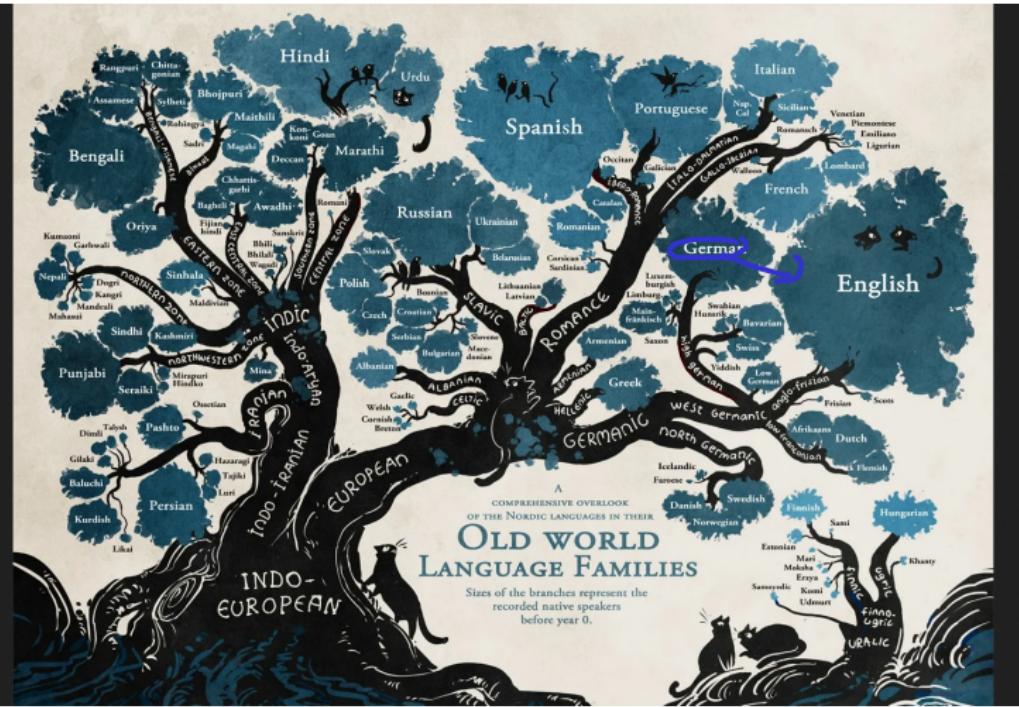


Figura: Evolucion de pictogramas a letras, fuente:
<https://usefulcharts.com/blogs/charts/evolution-of-the-english-alphabet>



Lenguajes de programación

La **programación** se define como una actividad general del hombre, que significa la acción de extender o cambiar la funcionalidad de un sistema[VanRoy].

- Programar es decirle a un computador (o a alguna máquina) como realizar su trabajo.
- La programación es una actividad de amplio espectro
- La programación de sistemas de software consta de dos partes esenciales: la ciencia y la tecnología.

Formalismos
Lógica etc

Lenguajes de programación

- Un **lenguaje de programación** es un lenguaje artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por una máquina.
- Conjunto de símbolos y reglas sintácticas y semánticas.

Contenido

1 Conceptos

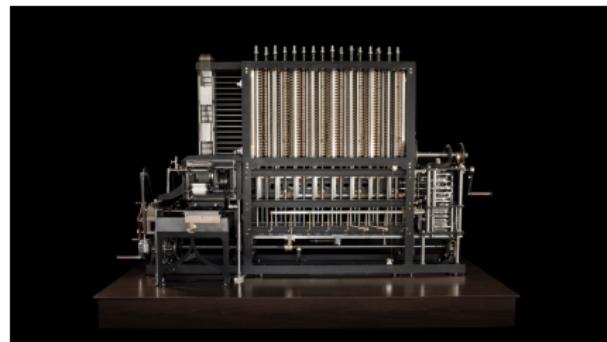
2 Un poco de historia

3 Perspectiva de los paradigmas de programación

4 Motivación del curso

Historia de los lenguajes de programación I

- Charles Babagge (Máquina Analítica) y Ada Lovelace (Primera programadora) (Mediados del siglo XIX).



Historia de los lenguajes de programación II

■ Programa escrito por Ada Lovelace

Number of Operations.	Nature of Operations.	Variables used or opn.	Variables receiving results.	Indication of change in the value on any Variable.	Data.		Working Variables.										Result Variables.				
					Statement of Results.		V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}	V_{13}	V_{14}	
1	\times	$V_2 \times V_3$	$V_2 \times V_3 \times V_4$	$(V_2 = V_2)$ $(V_3 = V_3)$ $(V_4 = V_4)$			2	a	2a	2a											
2	$-$	$V_4 - V_1$	$V_4 - V_1$	$(V_4 = V_4)$			2a	-1			1										
3	$+$	$V_4 + V_1$	$V_4 + V_1$	$(V_4 = V_4)$			2a+1				1										
4	$-$	$V_4 - V_2$	$V_4 - V_2$	$(V_4 = V_4)$			2a-1				1										
5	$+$	$V_4 + V_2$	$V_4 + V_2$	$(V_4 = V_4)$ $(V_2 = V_2)$			2a+1				1										
6	$-$	$V_4 - V_3$	$V_4 - V_3$	$(V_4 = V_4)$ $(V_3 = V_3)$			2a-1				1										
7	$-$	$V_4 - V_1$	$V_4 - V_1$	$(V_4 = V_4)$ $(V_1 = V_1)$			a-1 (= 3)				1										
8	$+$	$V_4 + V_1$	$V_4 + V_1$	$(V_4 = V_4)$ $(V_1 = V_1)$			2				2										
9	$-$	$V_4 - V_2$	$V_4 - V_2$	$(V_4 = V_4)$ $(V_2 = V_2)$			2a				2										
10	\times	$V_{12} \times V_{13}$	$V_{12} \times V_{13}$	$(V_{12} = V_{12})$ $(V_{13} = V_{13})$			2a				2										
11	$+$	$V_{12} + V_{13}$	$V_{12} + V_{13}$	$(V_{12} = V_{12})$ $(V_{13} = V_{13})$			2a				2										
12	$-$	$V_{12} - V_{13}$	$V_{12} - V_{13}$	$(V_{12} = V_{12})$ $(V_{13} = V_{13})$			a-2 (= 2)				1										
13	$-$	$V_4 - V_1$	$V_4 - V_1$	$(V_4 = V_4)$ $(V_1 = V_1)$			2a-1				1										
14	$+$	$V_4 + V_1$	$V_4 + V_1$	$(V_4 = V_4)$ $(V_1 = V_1)$			2a+1-3				1										
15	$-$	$V_4 - V_2$	$V_4 - V_2$	$(V_4 = V_4)$ $(V_2 = V_2)$			2a-1				1										
16	\times	$V_2 \times V_3 \times V_4$	$V_2 \times V_3 \times V_4$	$(V_2 = V_2)$ $(V_3 = V_3)$ $(V_4 = V_4)$			2				2a-1										
17	$-$	$V_4 - V_1$	$V_4 - V_1$	$(V_4 = V_4)$ $(V_1 = V_1)$			2a-2				1										
18	$+$	$V_4 + V_1$	$V_4 + V_1$	$(V_4 = V_4)$ $(V_1 = V_1)$			2a-1-4				1										
19	$-$	$V_4 - V_3$	$V_4 - V_3$	$(V_4 = V_4)$ $(V_3 = V_3)$			2a-2				1										
20	\times	$V_2 \times V_3 \times V_4$	$V_2 \times V_3 \times V_4$	$(V_2 = V_2)$ $(V_3 = V_3)$ $(V_4 = V_4)$			2a-2-2				1										
21	\times	$V_{12} \times V_{13}$	$V_{12} \times V_{13}$	$(V_{12} = V_{12})$ $(V_{13} = V_{13})$			2a-2-2				1										
22	$+$	$V_4 + V_1$	$V_4 + V_1$	$(V_4 = V_4)$ $(V_1 = V_1)$			2a-2-2				1										
23	$-$	$V_4 - V_1$	$V_4 - V_1$	$(V_4 = V_4)$ $(V_1 = V_1)$			a-3 (= 1)				1										
24	$+$	$V_{12} + V_{13}$	$V_{12} + V_{13}$	$(V_{12} = V_{12})$ $(V_{13} = V_{13})$			$B_2 =$ $(V_2 = V_2)$				1										
25	$+$	$V_4 + V_1$	$V_4 + V_1$	$(V_4 = V_4)$ $(V_1 = V_1)$			$n-1 = 4+1=5$ by a Variable const.				1										

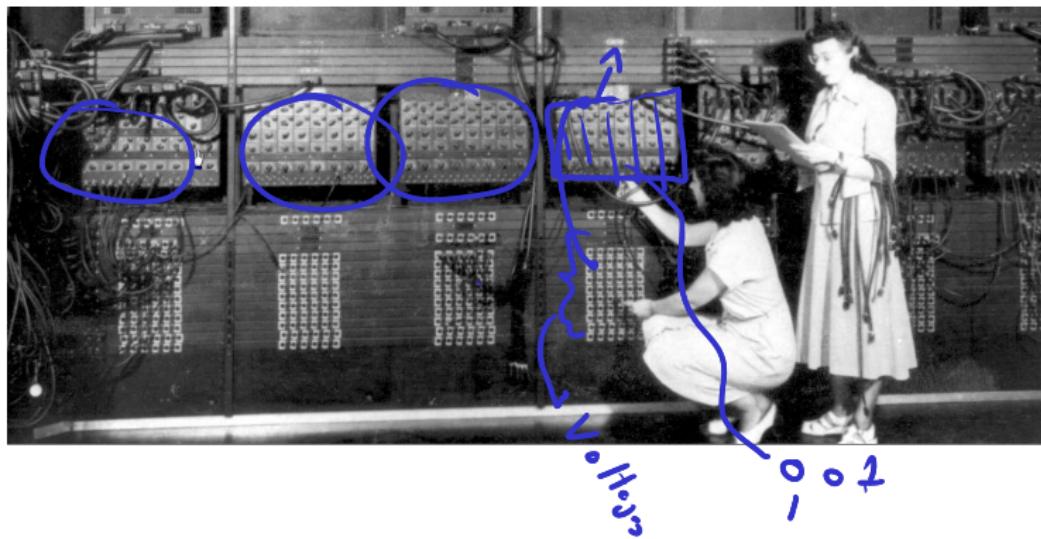
Here follows a repetition of Operations thirteen to twenty-three.

Historia de los lenguajes de programación III

- Herman Hollerith se percató de que podía codificar la información en tarjetas perforadas cuando observó a los conductores de trenes que identificaban a los pasajeros según el orificio que hacían en su respectivo ticket. En 1890 Hollerith codificó los datos del censo en tarjetas perforadas.
- En la década de 1920 los cálculos numéricos estaban basados en los números decimales. Con el paso del tiempo, se dieron cuenta de que la lógica podía ser representada con números, no sólo con palabras. **con número binarios**

Historia de los lenguajes de programación IV

- En 1943 se crea el sistema de codificación de ENIAC la primera computadora de propósito general.



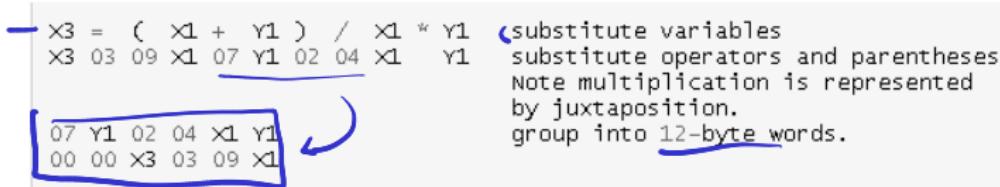
Historia de los lenguajes de programación V

-245C	-G ₁	-	-G ₁	-	1	100011010	
-2526					2	01011110	
-265C	b ₁				3	01011010	
-2727			b ₁	b ₁	4	11011110	
-235C	a	T _{n+1}	-b _n	b _n	5	11101010	
Subr. 27	a=6.				6	110111001	
Test					7	-011	
Add 20t6l					8	00101100	
Subr. 26	T _n				9	010111001	
c 5 25		T _n			10	10011110	
-255C					11	10011010	
Test					12	-011	
Stop	0	0	-b _n	b _n	13	111	
-265C	b _n	T _n	-b _n	b _n	14	01011010	
Subr. 21	b _{n+1}				15	10101001	
c 5 27	b _{n+1}		b _{n+1}		16	11011110	
-275C	-b _{n+1}				17	11011010	
c 5 26					18	01011110	
22t6l.		T _n	-b _{n+1}	b _{n+1}	19	01101000	
20	-3	10111 etc			23	-a	init.
21					25	-	final

carga 1
R1
Supr
R2 y R2
R3

Historia de los lenguajes de programación VI

- Short Code de John Mauchly en 1949 (BINAC y UNIVAC I). Por ejemplo la expresión: $a = \frac{b+c}{b*c}$ se computa así:



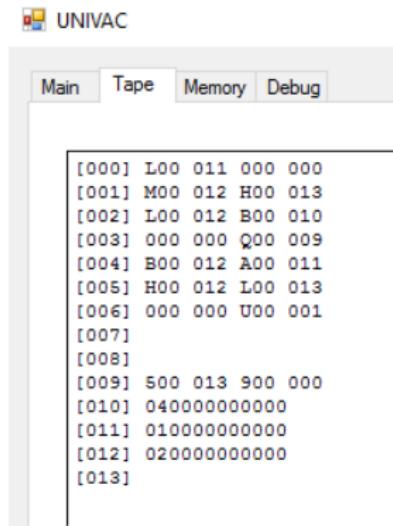
X3 = (x1 + y1) / x1 * y1
X3 03 09 x1 07 y1 02 04 x1 y1

07 Y1 02 04 x1 y1
00 00 X3 03 09 x1

substitute variables
substitute operators and parentheses
Note multiplication is represented by juxtaposition.
group into 12-byte words.

- A-0, A-1, A-2 entre 1951 y 1953 (UNIVAC). El siguiente programa calcula el factorial de un número almacenado en un registro.

Historia de los lenguajes de programación VII



The screenshot shows a window titled "UNIVAC" with tabs for Main, Tape, Memory, and Debug. The Main tab is selected, displaying a memory dump. The data is organized into two columns: address and binary value. The addresses range from [000] to [013]. The binary values represent various instruction and data types typical of early computer systems.

Address	Value
[000]	L00 011 000 000
[001]	M00 012 H00 013
[002]	L00 012 B00 010
[003]	000 000 Q00 009
[004]	B00 012 A00 011
[005]	H00 012 L00 013
[006]	000 000 U00 001
[007]	
[008]	
[009]	500 013 900 000
[010]	0400000000000
[011]	0100000000000
[012]	0200000000000
[013]	

Mayor información en:

<http://museo.inf.upv.es/en/univac/> así mismo se tiene un simulador disponible en:

<http://www.historicsimulations.com/univac.html>

Historia de los lenguajes de programación VIII

- Fortran (FORmula TRANslator) por John Backus et al. en 1953.

```
C AREA OF A TRIANGLE - HERON'S FORMULA
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      INTEGER A,B,C
      READ(5,501) A,B,C
501 FORMAT(3I5)
      IF(A.EQ.0 .OR. B.EQ.0 .OR. C.EQ.0) STOP 1
      S = (A + B + C) / 2.0
      AREA = SQRT( S * (S - A) * (S - B) * (S - C) )
      WRITE(6,601) A,B,C,AREA
601 FORMAT(4H A= ,15,5H B= ,15,5H C= ,15,8H AREA= ,F10.2,
$13H SQUARE UNITS)
      STOP
      END
```

- LISP (LISt Processor) por John McCarthy et al. en 1958.

```
(defun averagenum (n1 n2 n3 n4)
  (/ (+ n1 n2 n3 n4) 4)
)
(write(averagenum 10 20 30 40))
```



Historia de los lenguajes de programación IX

- COBOL (COmmon Business Oriented Language) por Grace Hopper en 1959.

```
DATA DIVISION.  
  
WORKING-STORAGE SECTION.  
01 Num1                      PIC 9  VALUE ZEROS.  
01 Num2                      PIC 9  VALUE ZEROS.  
01 Result                     PIC 99 VALUE ZEROS.  
  
PROCEDURE DIVISION.  
    DISPLAY "Enter first number (1 digit) : " WITH NO ADVANCING.  
    ACCEPT Num1.  
    DISPLAY "Enter second number (1 digit) : " WITH NO ADVANCING.  
    ACCEPT Num2.  
    MULTIPLY Num1 BY Num2 GIVING Result.  
    DISPLAY "Result is = ", Result.  
    STOP RUN.
```



Historia de los lenguajes de programación X

■ ALGOL (ALGOrithmic Language) 60 en 1960.

```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;
begin
    integer p, q;
    y := 0; i := k := 1;
    for p := 1 step 1 until n do
        for q := 1 step 1 until m do
            if abs(a[p, q]) > y then
                begin y := abs(a[p, q]);
                i := p; k := q
                end
end Absmax
```

Historia de los lenguajes de programación XI

- APL (A Programming Language) por Kenneth Iverson(IBM)
<https://tryapl.org/>
- Simula por Ole Johan Dahl y Kristen Nygaard y SNOBOL
(StriNg Oriented symBOlic Language) por los Laboratorios Bell en 1962. Este es el primer lenguaje orientado a objetos

```
Begin
    OutText ("Hello, World!");
    OutImage;
End;
Begin
    Class Glyph;
        Virtual: Procedure print Is Procedure print;;
    Begin
End;
```

- CPL (Combined Programming Languaje) en 1963. Está basado en Algol y simplifica la creación de programas en computación científica.

```
rec function Fact1[x] = (x = 0) -> 1, xFact1[x - 1]
```



Historia de los lenguajes de programación XII

~B

- BASIC (Beginners All-purpose Symbolic Instruction Code) por Kurtz y PL/1 (*Programming Language 1*) de IBM en 1964.

```
PROGRAM series
! add the first 100 terms of a simple series
! True BASIC automatically initializes variables to zero,
! but other languages might not.
LET sum = 0
FOR n = 1 to 100
    LET sum = sum + 1/(n*n)
    PRINT n,sum
NEXT n
END
```

Basic
C DOS

Ejemplo en PL/1

```
Test: procedure options(main);

declare My_String char(20) varying initialize('Hola, mundo!');

put skip list(My_String);

end Test;
```

Historia de los lenguajes de programación XIII

- BCPL (Basic Combined Programming Language) por Martin Richards en 1967.
- ALGOL 68 y Logo por Danny Bobrow, Wally Feurzeig y Seymour Papert en 1968. Logo es un lenguaje funcional y estructurado, de muy fácil aprendizaje. Se utiliza para la enseñanza de la programación en niños y jóvenes.

<https://www.calormen.com/jslogo/>

-
- C por Dennis Ritchie y Ken Thompson entre 1969 y 1973. Influido por Algol 68 (Lenguaje A) y BCPL (Lenguaje B).
 - Pascal por Wirth y SmallTalk en 1970. Pascal es estáticamente y fuertemente tipado. SmallTalk es de tipado dinámico.

```
Program Lesson1_Program1;
Begin
    Write('Hello World. Prepare to learn PASCAL!!');
    Readln;
End.
```

Historia de los lenguajes de programación XIV

```
'Hello World!' displayNL  
1 to: 20 do: [:x | x printNL] !  
  
x := Array new: 20.  
x printNL.  
(x at: 1) printNL.  
x at: 1 put: 99 .  
(x at: 1) printNL .
```

- Prolog (PROgrammation en LOGique) por Colmerauer, Roussel, y Kowalski en 1972.

- Base del conocimiento:

likes(mary, food).
likes(mary, wine).
likes(john, wine).
likes(john, mary).

$P(x, y) \Leftrightarrow x \text{ le gusta } y$

- Consultas:

Historia de los lenguajes de programación XV

```
likes(mary, food).
Yes
likes(john, food).
No
likes(X, wine).
mary
john
No
```

- ML (Meta Language) por Robin Milner en 1973.

```
fun reverse [] = []
| reverse (h :: t) = reverse t @ [h];

fun concat_space s = s ^ " ";

(* Prints each command line arg, suffixed with a space. *)

val _ =
  let
    val args = CommandLine.arguments()
  in
    map (print o concat_space) (reverse args);
    print "\n"
  end;
```

Historia de los lenguajes de programación XVI

- Scheme por Guy L. Steele y Gerald Jay Sussman en 1975.
- SQL (Structured Query Language) en 1978.
- Ada por Jean Ichbiah et al. en 1983.
- C++ por Bjarne Stroustrup en 1983. *C + objetos*
- Common Lisp en 1984.
- Eiffel, Erlang, Perl, Tcl y Fl a finales de los 80's. Ejemplo Erlang:

```
sort([Pivot|T]) ->
    sort([X || X <- T, X < Pivot]) ++
    [Pivot] ++
    sort([X || X <- T, X >= Pivot]);
sort([]) -> [].
```

- Haskell (en honor a Haskell Curry) en 1990.

Historia de los lenguajes de programación XVII

```
main = do
    forM_ [1..3] $ \i -> do
        print i

    forM_ [7..9] $ \j -> do
        print j

    withBreak $ \break ->
        forM_ [1..] $ \_ -> do
            p "loop"
            break ()

    where
    withBreak = ('runContT' `return) . callCC
    p = liftIO . putStrLn
```

- Python, Lua, Java, Delphi, JavaScript, PHP, Rebol, Visual Basic, Mozart, entre otros durante los años 90's.
- C#, .NET, J#, Scala, Factor, entre otros apartir del año 2000.

Historia de los lenguajes de programación XVII

```
main = do
    forM_ [1..3] $ \i -> do
        print i

    forM_ [7..9] $ \j -> do
        print j

    withBreak $ \break ->
        forM_ [1..] $ \_ -> do
            p "loop"
            break ()

    where
    withBreak = ('runContT' return) . callCC
    p = liftIO . putStrLn
```

- Python, Lua, Java, Delphi, JavaScript, PHP, Rebol, Visual Basic, Mozart, entre otros durante los años 90's.
- C#, .NET, J#, Scala, Factor, entre otros apartir del año 2000.

Historia de los lenguajes de programación XVII

```
main = do
    forM_ [1..3] $ \i -> do
        print i

    forM_ [7..9] $ \j -> do
        print j

    withBreak $ \break ->
        forM_ [1..] $ \_ -> do
            p "loop"
            break ()

    where
    withBreak = ('runContT' return) . callCC
    p = liftIO . putStrLn
```

for (int i = 0....
System.out.

5.times(puts(..))

hola.charAt(2)

- Python, Lua, Java, Delphi, JavaScript, PHP, Rebol, Visual Basic, Mozart, entre otros durante los años 90's.
- C#, .NET, J#, Scala, Factor, entre otros apartir del año 2000.

Contenido

- 1 Conceptos**
- 2 Un poco de historia**
- 3 Perspectiva de los paradigmas de programación**
- 4 Motivación del curso**

Paradigmas de programación

Los principales paradigmas de programación son:

- Declarativos (Funcional, Lógico, Por Restricciones)
- Imperativo
- Relacional
- Orientado a Objetos
- Por Restricciones
- Concurrente
- Orientado a aspectos
- Orientado a agentes

Paradigma programación Declarativa

- Una operación es declarativa si siempre que es llamada con los mismos argumentos retorna el mismo resultado.
- Una operación declarativa es:

- ↳ ■ **Independiente:** depende solo de sus argumentos
- ↳ ■ **Sin estado:** no hay memoria entre distintos llamados
- **Determinista:** un llamado con los mismos argumentos da siempre el mismo resultado, sin importar desde el lugar que se haga.

- ↳ ■ Ejemplo: HTML, XML, CSS, Mercury, Prolog.


```

## SQL



```
SELECT * FROM Users WHERE Country='Mexico';
```

1

100

## XML

```
<article>
 <header>
 <title>Programación declarativa</title>
 <text>Solo escribe sin preocuparte de más :)</text>
 </header>
</article>
```

JSON { 'nombre': 'Juan', 'edad': 20 }

# Paradigma programación Funcional

$F(x, y; x)$

$\text{fun } h(x)$

- Basado en el cálculo  $\lambda$ .

- Manejo implícito de la memoria.

- El concepto de función es fundamental.



- Funciones son ciudadanos de primera clase (las funciones pueden ser parámetros o valores de retorno de otras funciones).

$\text{fun } F(x, y)$



- Programa: Conjunto de funciones + Aplicación.

- Ejemplos: Lisp, Haskell, Scheme, ML.

$F(3, h)$

# Paradigma programación declarativo funcional

## Cálculo λ

- Diseñado para investigar la definición de función, la noción de aplicación de funciones y la recursión.
- Utilizado para definir algoritmos computables o decidibles.
- Es una estrategia para definir si un algoritmo es computable, ya que se ha demostrado que el problema de la parada es un problema indecidible.
- Cualquier función computable puede ser expresada y evaluada a través de este cálculo.
- Las funciones son consideradas un valor tipo procedimiento.

# Paradigma de Programación declarativo lógico

- Basado en el cálculo de predicados.
- Mecanismo de demostración automática de teoremas.
- Esencial: Concepto de deducción lógica.
- Programa: Conjunto de axiomas y un objetivo.
- Ejemplos: Prolog.

# Paradigma programación Imperativa

- Orientado por la máquina.
- Alto nivel.
- Esencial: Asignación y secuenciación.
- La programación está dada en términos del estado del programa.
- Programa: Secuencia de instrucciones.
- Ejemplos: Fortran, Algol, Basic, C, Pascal.

$V = 3$  ↘  
|  
|  
|  
 $V = 8$  ↘

# Paradigma programación Imperativa

## JavaScript

```
class Number {
 constructor (number = 0) {
 this.number = number;
 }

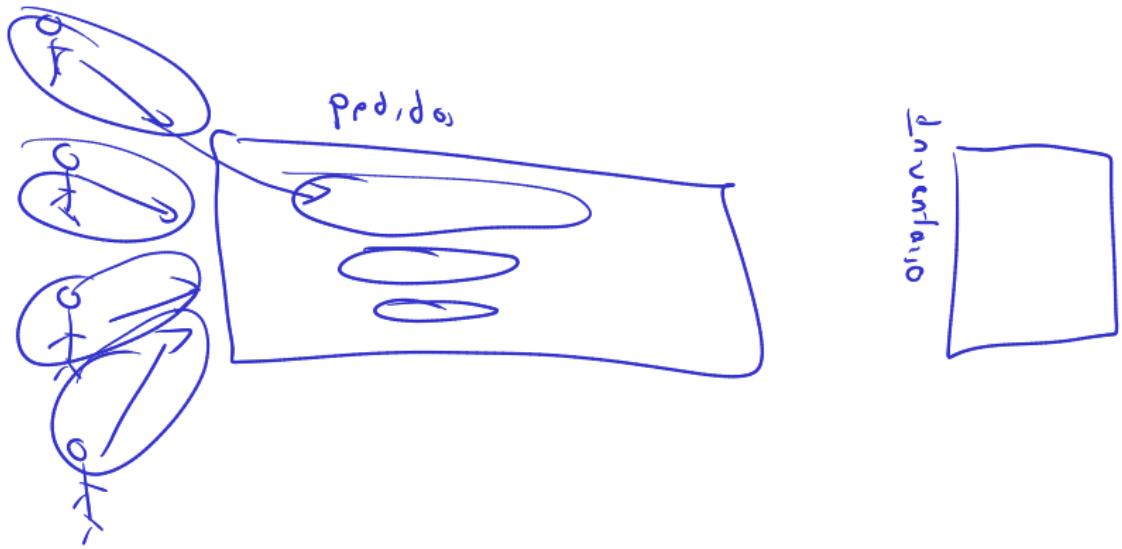
 add (x) {
 this.number = this.number + x;
 }
}
const myNumber = new Number (5);
myNumber.add(3);
alert (myNumber.number); // 8
```

# Paradigma de Programación Orientada a Objetos

- Se representa el mundo real mediante objetos y sus interacciones.
- Basado en el concepto computacional de objeto.
- Esencial: Concepto de objeto, herencia, mensaje.
- Programa: Conjunto de objetos y sus interacciones.
- Ejemplos: Smalltalk, Java, C++, Obliq, etc.

# Paradigma de Programación Concurrente

- Basado en la teoría de concurrencia y cálculos de procesos (Cálculo  $\pi$ , CCS, CCP).
- Esencial: Mecanismos de comunicación entre procesos.
- Programa: Conjunto de procesos.
- Ejemplos: PICT, MWB, Erlang.
- Este paradigma funciona bien en lenguajes funcionales, ya que no se requiere sincronización (semáforos).



# Paradigma de Programación Concurrente

## Ejemplo en Erlang

```
-module(ejemploFLP).

-export([iniciar/0, dialogo/2]).

dialogo(Entrada, 0) ->
 done;
dialogo(Entrada, Contador) ->
 io:format("~p~n", [Entrada]),
 dialogo(Entrada, Contador - 1).

iniciar() ->
 spawn(ejemploFLP, dialogo, [hola, 5]),
 spawn(ejemploFLP, dialogo, [adios, 4]).

%Funcionamiento
%c(ejemploFLP).
%ejemploFLP:iniciar().
```

# Paradigma de Programación Relacional

- Las funciones pueden tener cero, una o más salidas (Frente a funciones)
- Puede intercambiarse el rol de las entradas y salidas
- Selección no-determinista de una opción entre varias
- Ejemplo: Prolog (Búsqueda sobre una base de conocimiento), Analizadores sintácticos, Bases de datos relacionales (SQL).

# Paradigma de Programación Relacional

Ejemplo de paradigma relacional con Prolog.

```
%Base del conocimiento
amigo(juan, pedro).
amigo(juan, carlos).
amigo(pedro, maria).

%consultas
%amigo(juan, pedro). %Retorna yes
%amigo(juan, X). %Retorna X= pedro y X = carlos
```

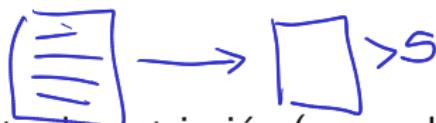
Observe que

- 1 Hay una búsqueda en un conjunto de datos
- 2 Se retornan cero o más valores en una consulta.

SQL también es un lenguaje relacional de búsqueda sobre bases de datos.

Select \* from paises where num\_hab > 5

# Paradigma de Programación por Restricciones



- Basado en el concepto de restricción (un predicado o relación lógica).
- **Esencial:** Concepto de consecuencia lógica.
- **Esencial:** Búsqueda en árboles y reducción de dominios (distribución y propagación).
- **Programa:** Variables + Restricciones (Conjunto de Relaciones entre variables) + Estrategia de exploración.
- La búsqueda de soluciones es concurrente (se exploran varias posibilidades a la vez).
- **Ejemplos:** CLP, Mozart.

# Paradigma de Programación por Restricciones

Ejemplo de paradigma de programación con restricciones con Mozart OZ. Problema, encontrar los valores de las letras para que  $\text{SEND} + \text{MORE} = \text{MONEY}$ .

```
declare
proc {Money Root}
local
 S E N D M O R Y
in
 Root = sol(s:S e:E n:N d:D m:M o:O r:R y:Y)
 Root ::: 0#9
 las variables entre 0 y 9
 %Restricciones
 {FD.distinct Root}
 %S y M distintos de 0 (si no todo se hace 0)
 S \=: 0
 M \=: 0
 1000*S + 100*E + 10*N + D
 +
 1000*M + 100*O + 10*R + E
 =: 10000*M + 1000*O + 100*N + 10*E + Y
 %Estrategia de búsqueda (probar primero con números pequeños del dominio)
 {FD.distribute ff Root}
end
%Explore todas las posibilidades
{ExploreAll Money}
```

% Registro con letras  
% Dominio de búsqueda de

Minizinc

# Paradigma de Programación Orientada a Aspectos

- Modularidad de las aplicaciones y separación de conceptos (generalmente conceptos técnicos y comunes a toda la aplicación).
- Separación de las funcionalidades comunes utilizadas en la aplicación de las funcionalidades propias de cada módulo.
- Esencial: Concepto de aspecto (funcionalidad transversal).
- Ejemplos: AspectJ, Aspect, phpAspect, Aspyct AOP.

# Paradigma de Programación Orientada a Agentes

- Se representa el mundo real mediante agentes y sus interacciones a través de mensajes.
- Basado en el concepto de agentes.
- Un agente es una entidad computacional situada en algún entorno y que es capaz de ejecutar acciones autónomas en dicho entorno con el fin de cumplir sus objetivos de diseño.
- Hilo de ejecución independiente, comunicación por paso de mensajes, conocimiento parcial del entorno, mecanismo de toma de decisiones, reactividad, proactividad, habilidad social.
- Programa: Conjunto de agentes y sus interacciones.
- Ejemplos: JADE, JASON.

# Algunos Retos

- Enfoques basados en componentes.
- Mecanismos de seguridad y de confiabilidad, seguridad en hilos.
- Énfasis en movilidad y distribución. Enfoques basados en paradigmas y tecnologías actuales (computación grid y cloud, Map and Reduce).

# Contenido

1 Conceptos

2 Un poco de historia

3 Perspectiva de los paradigmas de programación

4 Motivación del curso



# Por qué estudiar los conceptos de lenguajes de programación?

- Incrementa la capacidad para expresar ideas.
- Amplía el espectro de conocimientos necesario para seleccionar un lenguaje.
- Incrementa la habilidad para aprender nuevos lenguajes y paradigmas.

# Por qué estudiar los conceptos de lenguajes de programación?

- Mejor entendimiento de como los lenguajes de programación están implementados.
- Mejor uso de los lenguajes de programación que ya se conocen.
- Progreso global de las ciencias computacionales.

# Preguntas

?

# Próxima sesión

- Repaso de Gramáticas y Dr Racket.

