Fundamentos de lenguajes de programación

Robinson Duque, MEng - Ph.D

Universidad del Valle

robinson.duque@correounivalle.edu.co

Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Este documento es una adaptación del material original de los profesores
Carlos Andres Delgado y Carlos Alberto Ramírez

Overview

Lenguaje

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

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].

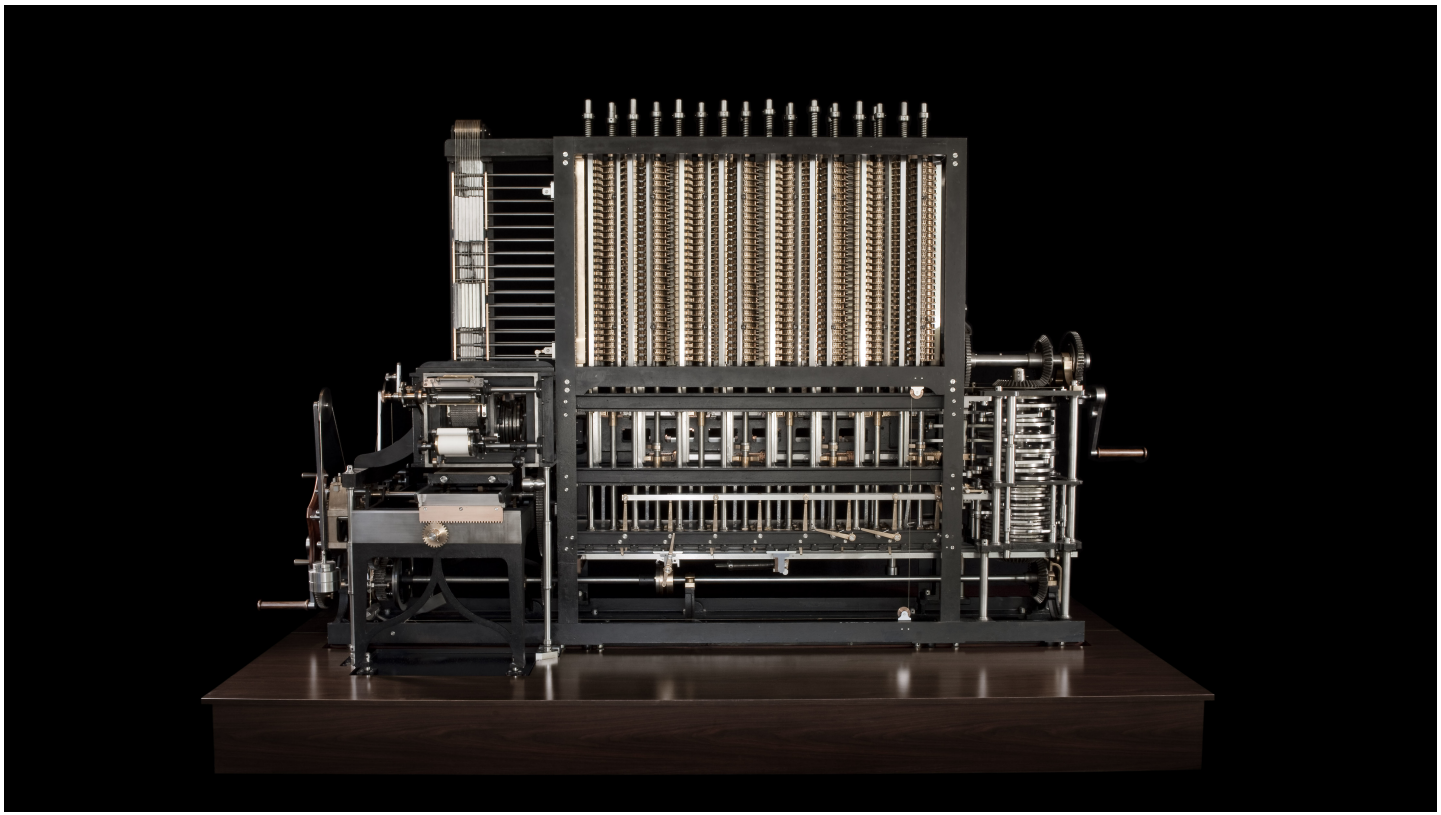
- Programar es decirle a un computador (o a alguna máquina) como realizar su trabajo.
- La programación es una actividad de amplio espectro realizada tanto por no especialistas como por especialistas.
- La programación de sistemas de software consta de dos partes esenciales: la ciencia y la tecnología.

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.

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

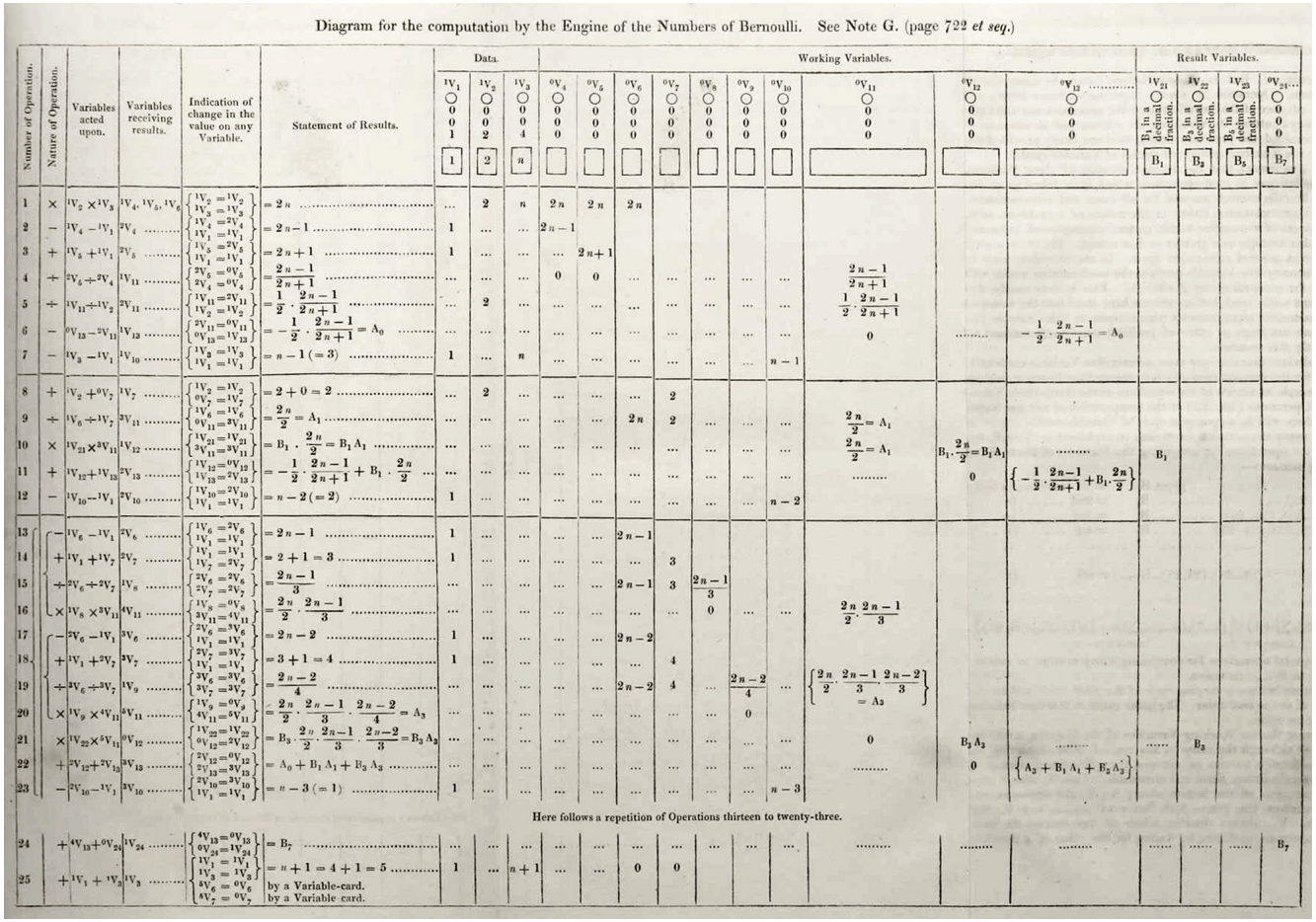


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

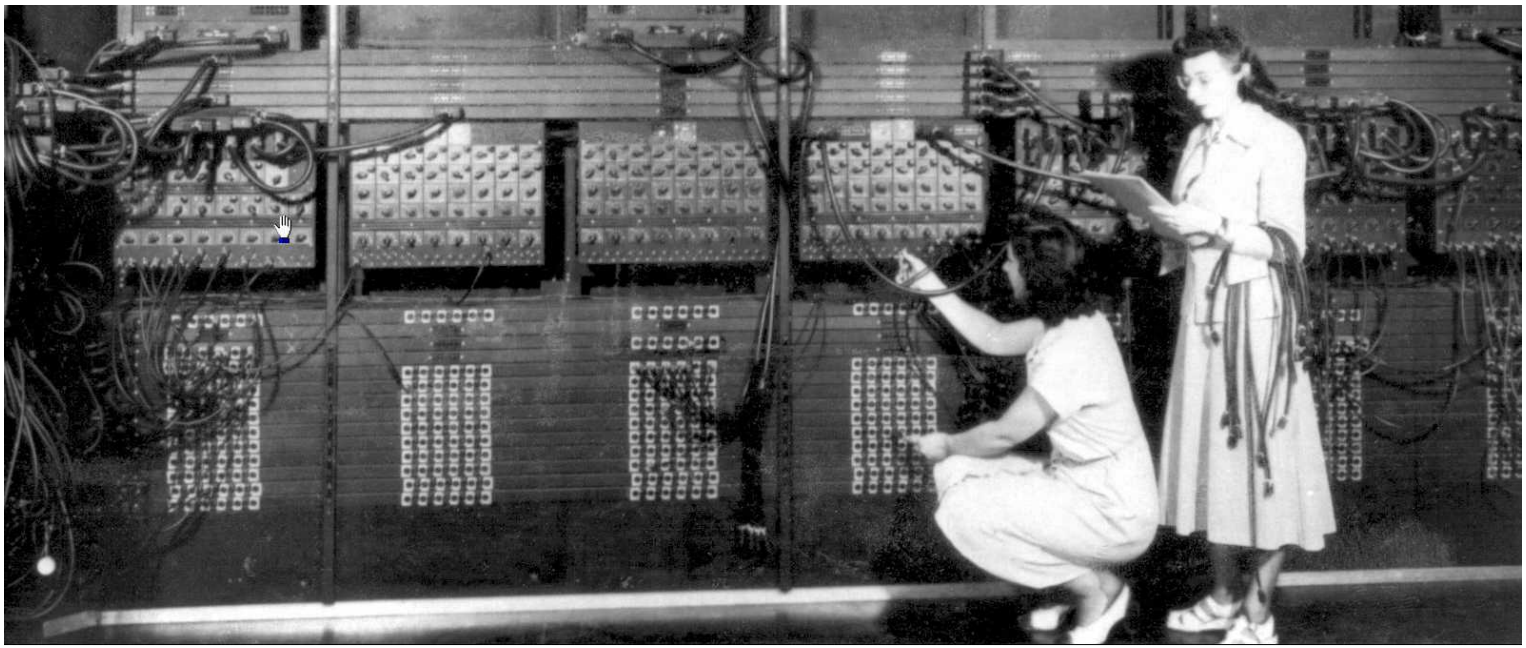
Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data. $^1V_1$	Data. $^1V_2$	Data. $^1V_3$	Working Variables. $^0V_4$	$^0V_5$	$^0V_6$	$^0V_7$	$^0V_8$	$^0V_9$	$^0V_{10}$	$^0V_{11}$	$^0V_{12}$	$^0V_{13}$	Result Variables. $^1V_{21}$	$^1V_{22}$	$^1V_{23}$	$^0V_{24}$...
						1	2	n											B_1	B_2	B_3	B_7
1	×	$^1V_2 \times ^1V_3$	$^1V_4, ^1V_5, ^1V_6$	$\begin{cases} ^1V_2 = ^1V_2 \\ ^1V_3 = ^1V_3 \end{cases}$	$= 2n$	...	2	n	$2n$	$2n$	$2n$											
2	−	$^1V_4 - ^1V_1$	$^2V_4$	$\begin{cases} ^1V_4 = ^2V_4 \\ ^1V_1 = ^1V_1 \end{cases}$	$= 2n-1$	1	...	...	$2n-1$													
3	+	$^1V_5 + ^1V_1$	$^2V_5$	$\begin{cases} ^1V_5 = ^2V_5 \\ ^1V_1 = ^1V_1 \end{cases}$	$= 2n+1$	1	...	...	...	$2n+1$												
4	+	$^2V_5 \div ^2V_4$	$^1V_{11}$	$\begin{cases} ^2V_5 = ^0V_5 \\ ^2V_4 = ^0V_4 \end{cases}$	$= \dfrac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	$\dfrac{2n-1}{2n+1}$						
5	÷	$^1V_{11} \div ^1V_2$	$^2V_{11}$	$\begin{cases} ^1V_{11} = ^2V_{11} \\ ^1V_2 = ^1V_2 \end{cases}$	$= \dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	$\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1}$						
6	−	$^0V_{13} - ^2V_{11}$	$^1V_{12}$	$\begin{cases} ^2V_{11} = ^0V_{11} \\ ^0V_{13} = ^1V_{13} \end{cases}$	$= -\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} = A_0$	...	...	...	...	...	...	...	...	...	...	0	.....	$-\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} = A_0$				
7	−	$^1V_3 - ^1V_1$	$^1V_{10}$	$\begin{cases} ^1V_3 = ^1V_3 \\ ^1V_1 = ^1V_1 \end{cases}$	$= n-1\ (=3)$	1	...	n	...	...	...	...	...	...	$n-1$							
8	+	$^1V_2 + ^0V_7$	$^1V_7$	$\begin{cases} ^1V_2 = ^1V_2 \\ ^0V_7 = ^1V_7 \end{cases}$	$= 2+0=2$	...	2	...	...	...	...	2										
9	÷	$^1V_6 \div ^1V_7$	$^2V_{11}$	$\begin{cases} ^1V_6 = ^1V_6 \\ ^0V_{11} = ^2V_{11} \end{cases}$	$= \dfrac{2n}{2} = A_1$	...	...	...	...	...	$2n$	2	...	...	...	$\dfrac{2n}{2} = A_1$						
10	×	$^1V_{21} \times ^3V_{11}$	$^1V_{12}$	$\begin{cases} ^1V_{21} = ^1V_{21} \\ ^3V_{11} = ^3V_{11} \end{cases}$	$= B_1 \cdot \dfrac{2n}{2} = B_1 A_1$	...	...	...	...	...	...	...	...	...	...	$\dfrac{2n}{2} = A_1$	$B_1 \cdot \dfrac{2n}{2} = B_1 A_1$	.....	B_1			
11	+	$^1V_{12} + ^1V_{13}$	$^2V_{13}$	$\begin{cases} ^1V_{12} = ^0V_{12} \\ ^1V_{13} = ^2V_{13} \end{cases}$	$= -\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} + B_1 \cdot \dfrac{2n}{2}$ ...	...	...	...	...	...	...	...	...	...	...	.....	0	$\left\{-\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} + B_1 \cdot \dfrac{2n}{2}\right\}$				
12	−	$^1V_{10} - ^1V_1$	$^2V_{10}$	$\begin{cases} ^1V_{10} = ^2V_{10} \\ ^1V_1 = ^1V_1 \end{cases}$	$= n-2\ (=2)$	1	...	...	...	...	...	...	...	...	$n-2$							
13	−	$^1V_6 - ^1V_1$	$^2V_6$	$\begin{cases} ^1V_6 = ^2V_6 \\ ^1V_1 = ^1V_1 \end{cases}$	$= 2n-1$	1	...	...	...	...	$2n-1$											
14	+	$^1V_1 + ^1V_7$	$^2V_7$	$\begin{cases} ^1V_1 = ^1V_1 \\ ^1V_7 = ^2V_7 \end{cases}$	$= 2+1=3$	1	...	...	...	...	...	3										
15	÷	$^2V_6 \div ^2V_7$	$^1V_8$	$\begin{cases} ^2V_6 = ^2V_6 \\ ^2V_7 = ^2V_7 \end{cases}$	$= \dfrac{2n-1}{3}$	...	...	...	...	...	$2n-1$	3	$\dfrac{2n-1}{3}$									
16	×	$^1V_8 \times ^3V_{11}$	$^4V_{11}$	$\begin{cases} ^1V_8 = ^0V_8 \\ ^3V_{11} = ^4V_{11} \end{cases}$	$= \dfrac{2n}{2} \cdot \dfrac{2n-1}{3}$	...	...	...	...	...	...	...	0	...	...	$\dfrac{2n}{2} \cdot \dfrac{2n-1}{3}$						
17	−	$^2V_6 - ^1V_1$	$^3V_6$	$\begin{cases} ^2V_6 = ^3V_6 \\ ^1V_1 = ^1V_1 \end{cases}$	$= 2n-2$	1	...	...	...	...	$2n-2$											
18	+	$^1V_1 + ^2V_7$	$^3V_7$	$\begin{cases} ^2V_7 = ^3V_7 \\ ^1V_1 = ^1V_1 \end{cases}$	$= 3+1=4$	1	...	...	...	...	...	4										
19	÷	$^3V_6 \div ^3V_7$	$^1V_9$	$\begin{cases} ^3V_6 = ^3V_6 \\ ^3V_7 = ^3V_7 \end{cases}$	$= \dfrac{2n-2}{4}$	...	...	...	...	...	$2n-2$	4	...	$\dfrac{2n-2}{4}$	...	$\left\{\dfrac{2n}{2} \cdot \dfrac{2n-1}{3} \cdot \dfrac{2n-2}{3}\right\}$ $= A_2$						
20	×	$^1V_9 \times ^4V_{11}$	$^5V_{11}$	$\begin{cases} ^1V_9 = ^0V_9 \\ ^4V_{11} = ^5V_{11} \end{cases}$	$= \dfrac{2n}{2} \cdot \dfrac{2n-1}{3} \cdot \dfrac{2n-2}{4} = A_3$ ...	...	...	...	...	...	...	...	...	0	...							
21	×	$^1V_{22} \times ^5V_{11}$	$^6V_{12}$	$\begin{cases} ^1V_{22} = ^1V_{22} \\ ^0V_{12} = ^2V_{12} \end{cases}$	$= B_3 \cdot \dfrac{2n}{2} \cdot \dfrac{2n-1}{3} \cdot \dfrac{2n-2}{3} = B_3 A_2$ ...	...	...	...	...	...	...	...	...	...	...	0	$B_2 A_2$	.....	.....	B_3		
22	+	$^2V_{12} + ^2V_{13}$	$^3V_{13}$	$\begin{cases} ^2V_{12} = ^0V_{12} \\ ^2V_{13} = ^3V_{13} \end{cases}$	$= A_0 + B_1 A_1 + B_3 A_2$	...	...	...	...	...	...	...	...	...	...	.....	0	$\left\{A_2 + B_1 A_1 + B_2 A_2\right\}$				
23	−	$^2V_{10} - ^1V_1$	$^3V_{10}$	$\begin{cases} ^2V_{10} = ^3V_{10} \\ ^1V_1 = ^1V_1 \end{cases}$	$= n-3\ (=1)$	1	...	...	...	...	...	...	...	...	$n-3$							
Here follows a repetition of Operations thirteen to twenty-three.																						
24	+	$^4V_{13} + ^0V_{24}$	$^1V_{24}$	$\begin{cases} ^4V_{13} = ^0V_{13} \\ ^0V_{24} = ^1V_{24} \end{cases}$	$= B_7$	...	...	...	...	...	...	...	...	...	...	.....	.....	.....	.....	.....	.....	B_7
25	+	$^1V_1 + ^1V_3$	$^1V_3$	$\begin{cases} ^1V_1 = ^1V_1 \\ ^1V_3 = ^1V_3 \\ ^5V_6 = ^0V_6 \\ ^5V_7 = ^0V_7 \end{cases}$ by a Variable-card. by a Variable card.	$= n+1 = 4+1 = 5$	1	...	$n+1$	...	...	0	0										

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.

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

- 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   substitute variables
X3 03 09 X1 07 Y1 02 04 X1   Y1   substitute operators and parentheses
                                  Note multiplication is represented
                                  by juxtaposition.
07 Y1 02 04 X1 Y1                 group into 12-byte words.
00 00 X3 03 09 X1
```

- A-0, A-1, A-2 entre 1951 y 1953 (UNIVAC).

Historia de los lenguajes de programación VI

- 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= ,I5,5H  B= ,I5,5H  C= ,I5,8H  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 VII

- 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 VIII

- 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 IX

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

Historia de los lenguajes de programación X

- BCPL (Basic Combined Programming Language) por Martin Richards en 1967.
- ALGOL 68 y Logo por Danny Bobrow, Wally Feurzeig y Seymour Papert en 1968.
- C por Dennis Ritchie y Ken Thompson entre 1969 y 1973.
- Pascal por Wirth y SmallTalk en 1970.

Historia de los lenguajes de programación XI

- 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).
```

  - Consultas:

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

Historia de los lenguajes de programación XII

- 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;
```

- 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.

Historia de los lenguajes de programación XIII

- Common Lisp en 1984.
- Eiffel, Erlang, Perl, Tcl y FI 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.

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

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

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

Historia de los lenguajes de programación XIV

```
    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.

Paradigmas de programación - Conceptos generales

Un paradigma es un enfoque para programar máquinas (computadores) basado en un conjunto coherente de principios o teoría matemática [P. Van Roy]:

- Las teorías de computación resultan en diferentes paradigmas (λ calculus, π calculus, lógica de primer orden, etc)
- Ninguna teoría existente cubre todos los conceptos de programación

¿Porqué necesitamos tantos paradigmas? Para solucionar problemas más fácilmente utilizando el paradigma correcto!

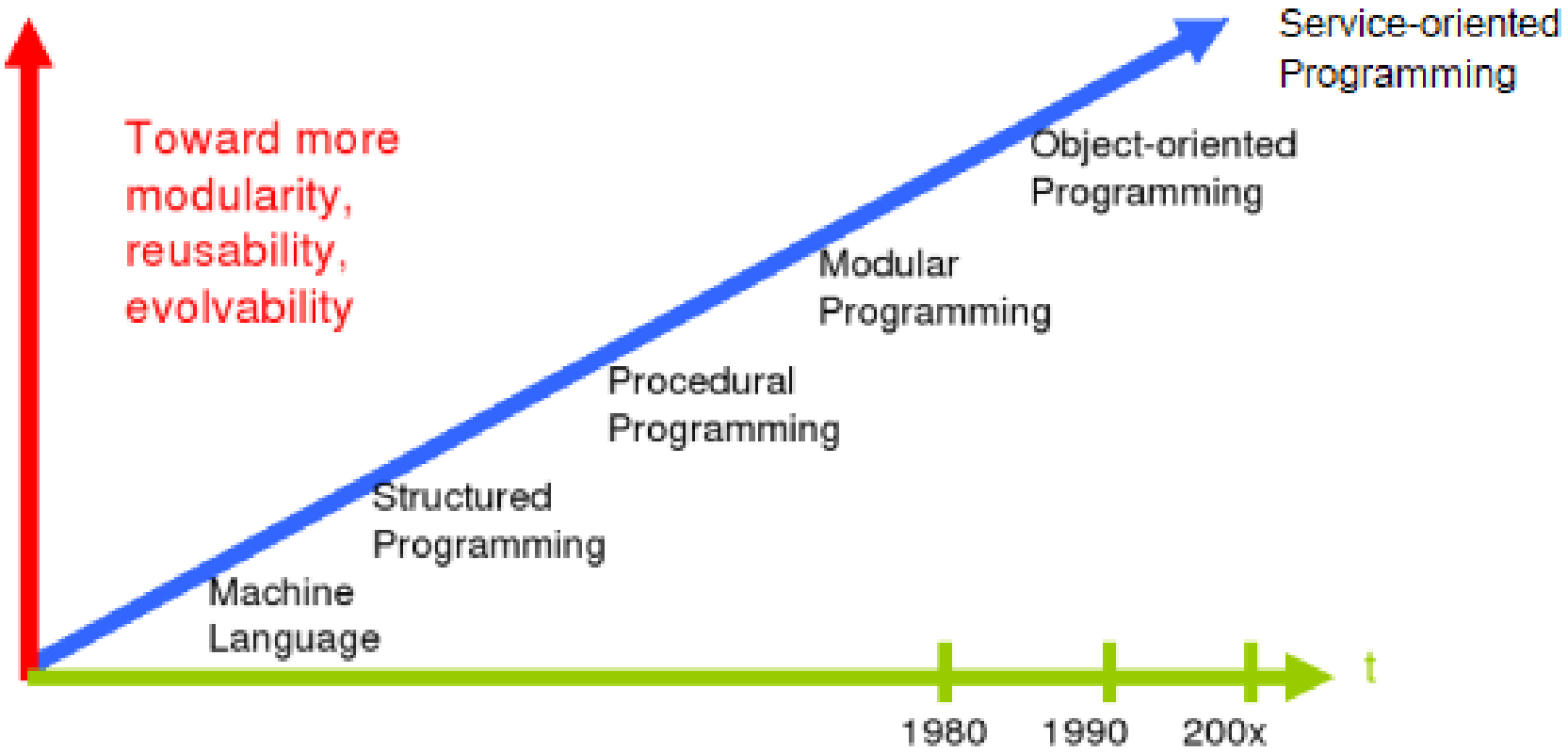
Paradigmas de programación - Conceptos generales

Un paradigma es un enfoque para programar máquinas (computadores) basado en un conjunto coherente de principios o teoría matemática [P. Van Roy]:

- Las teorías de computación resultan en diferentes paradigmas (λ calculus, π calculus, lógica de primer orden, etc)
- Ninguna teoría existente cubre todos los conceptos de programación

¿Porqué necesitamos tantos paradigmas? Para solucionar problemas más fácilmente utilizando el paradigma correcto!

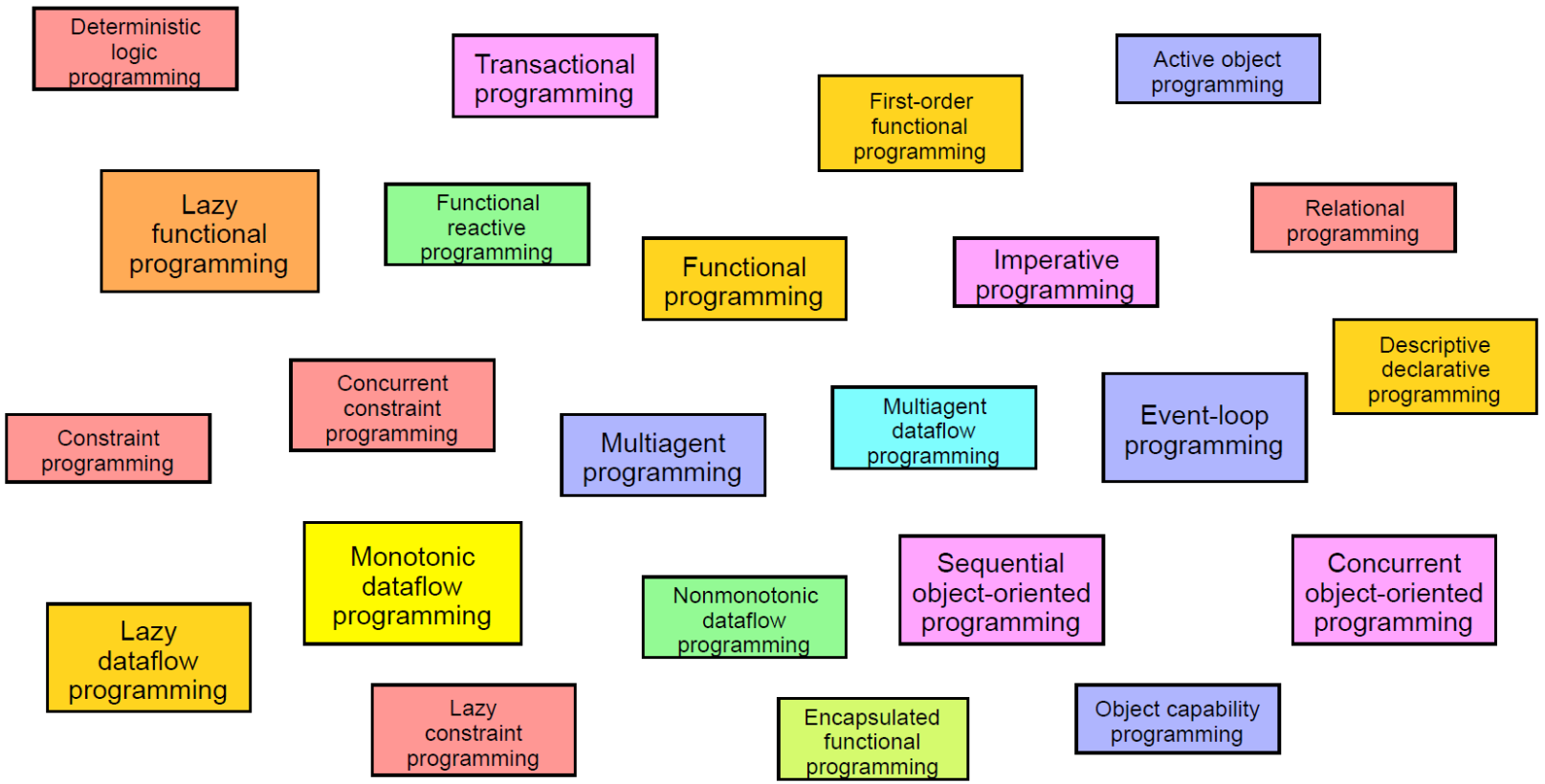
Paradigmas de programación - Conceptos generales



- ilustración típica de cómo ha evolucionado la programación
- el diagrama sólo muestra una pequeña parte; deja por fuera muchas ideas importantes

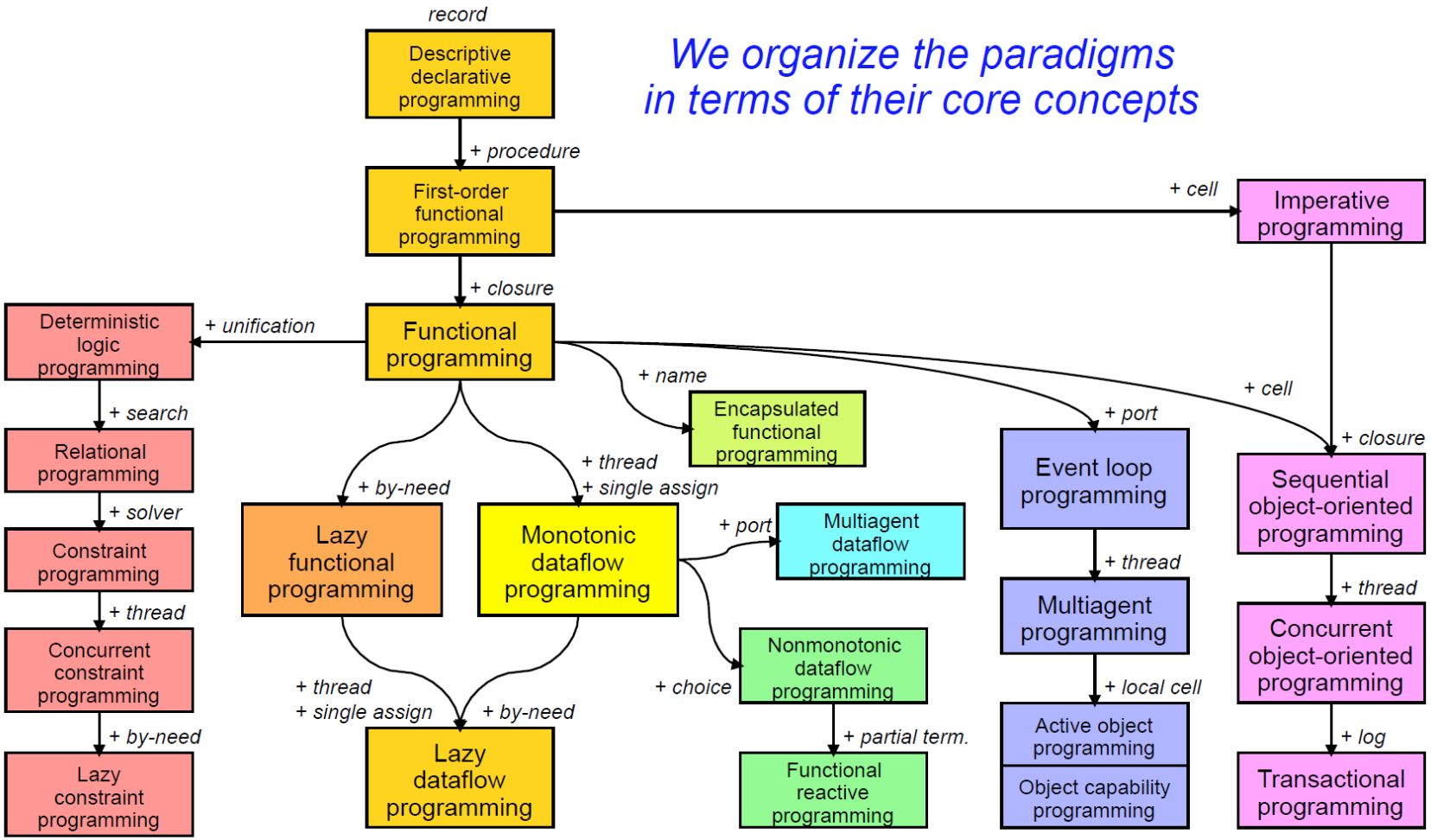
Taxonomía de los paradigmas de programación

La jungla de paradigmas de programación



Taxonomía de los paradigmas de programación

Taxonomía de paradigmas de programación



Paradigmas de programación

Los principales paradigmas de programación son:

- Declarativos (Funcional, Lógico, Por Restricciones)
- Imperativo
- Relacional
- Orientado a Objetos
- Por Restricciones
- Concurrente
- 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)
- Ejemplo: HTML, XML, CSS, Mercury, Prolog.

Paradigma programación Declarativa

HTML

```
<img src="./image.jpg" />
```

SQL

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

XML

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

Paradigma programación Funcional

- Basado en el cálculo λ (sistema formal 1930).
- El concepto de función es fundamental.
- Funciones son ciudadanos de primera clase (las funciones pueden ser parámetros o valores de retorno de otras funciones).
- Programa: Conjunto de funciones + Aplicación.
- Ejemplos: Lisp, Haskell, Scheme, ML.

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.

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 π, 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 relaciones 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 (ver ejemplo online).

```
%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.

Paradigma de Programación por Restricciones

- Basado en el concepto de restricción (un predicado o relación lógica).
- **Esencial:** Concepto de consecuencia lógica.
- **Esencial:** Búsqueda en arboles 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, MiniZinc.

Paradigma de Programación por Restricciones

Ejemplo: encontrar los valores de las letras para que SEND + MORE = MONEY (ver ejemplos online).

```
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)        % Registro con letras
   Root ::: 0#9                                       % Dominio de búsqueda de
        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}
```

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 fín 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.

¿Qué paradigma es el mejor?

- Cada uno es mejor para una clase específica de problema (la paradoja del paradigma - *"more is not better or worse, only different"*)
- Las fronteras de los paradigmas son completamente artificiales (sólo existen por razones históricas)
  - Java es sólo orientado a objetos (errado)
  - Scala es funcional, orientado a objetos, basado en actores (correcto)
- Un programa grande casi siempre necesita diversos paradigmas (por esto es necesario aprender múltiples paradigmas)
- Un buen lenguaje debería soportar diversos paradigmas (Scala y Erlang se mueven en la dirección correcta; Java y C++ se están estancando..)

¿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 Racket.