Clase 2
Lenguajes de Programación
IIC1005
2019

Dr. Luis Ramírez Donoso

llramirez@ing.puc.cl

Hoy: Lenguajes de Programación

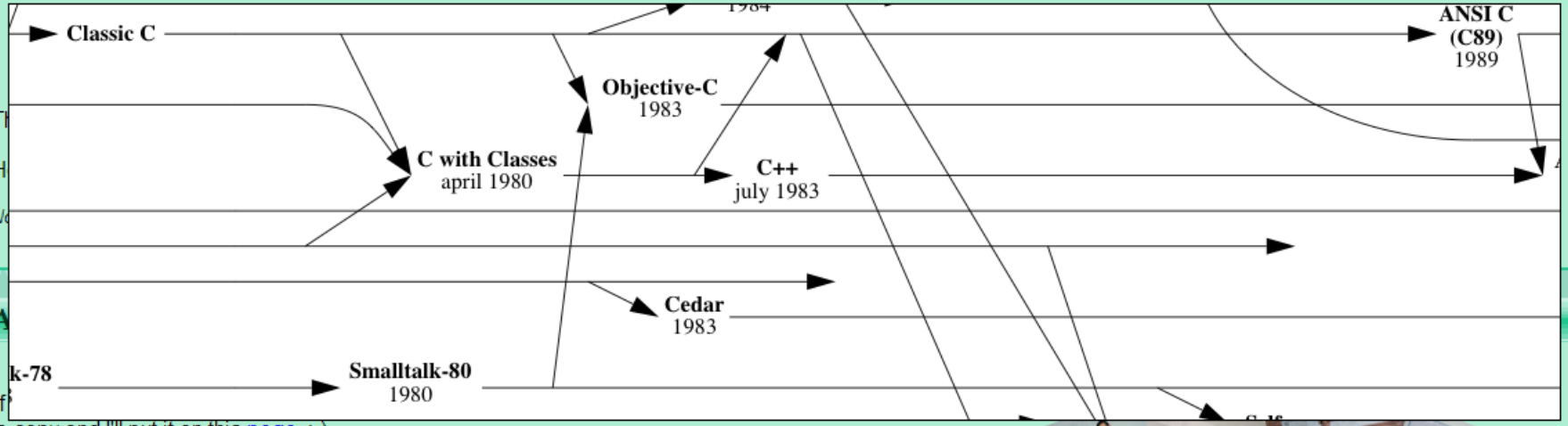
- Algo de Historia, y luego …
  - Assembly
  - Fortran
  - C
  - Prolog
  - C++
  - Java / C#
  - Objective-C
  - Python
  - Ruby
  - PHP
  - Javascript
  - R/Matlab (Computacion Cientifica)
  - SQL

Un poquito de historia

http://www.levenez.com/lang/

Below, you can see the preview of the **Computer Languages History** (move on the white zone to get a bigger image):

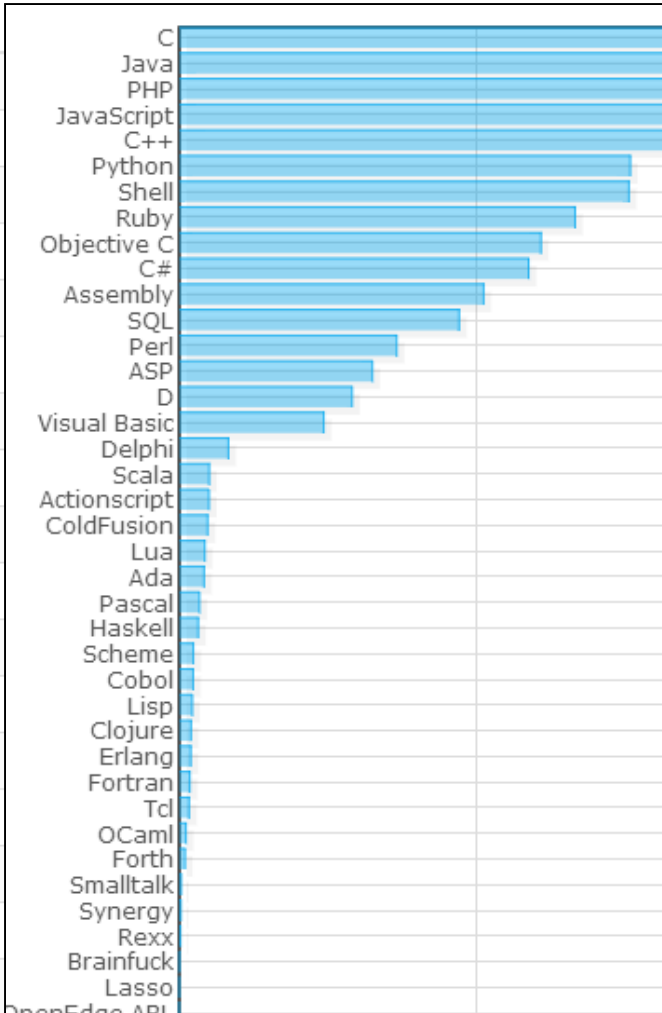
If you want to print this timeline, you can **freely** download one of the following PDF files:



a copy and I'll put it on this page. ;-)

TIOBE vs. LangPop

Oct 2015	Oct 2014	Change	Programming Language	Ratings	Change
1	2	↑	Java	19.543%	+6.04%
2	1	↓	C	16.190%	-1.47%
3	4	↑	C++	5.749%	+0.88%
4	5	↑	C#	4.825%	+0.08%
5	8	↑	Python	4.512%	+2.18%
6	7	↑	PHP	2.561%	-0.38%
7	13	⇈	Visual Basic .NET	2.462%	+0.71%
8	12	⇈	JavaScript	2.292%	+0.52%
9	9		Perl	2.247%	+0.13%
10	16	⇈	Ruby	1.825%	+0.70%
11	11		Delphi/Object Pascal	1.637%	-0.18%
12	31	⇈	Assembly language	1.573%	+1.16%
13	14	↑	Visual Basic	1.515%	-0.05%
14	3	⇊	Objective-C	1.419%	-8.68%
15	19	⇈	Swift	1.277%	+0.52%
16	20	⇈	Pascal	1.194%	+0.47%
17	27	⇈	MATLAB	1.159%	+0.55%



¿Cuánto importa la popularidad?

- Es importante, pero hay lenguajes que parecen poco populares y son muy usados en ciertas áreas:
- En Bancos y grandes compañías: COBOL
- Aplicaciones matemáticas: FORTRAN
- Etc…

ERLANG

- Primera versión de Erlang implementada en ProLog
- ¿Quién lo usa?
  - Amazon.com: Para su BD SimpleDB
  - Yahoo! : Para delicious. 5 millones de usuarios, 150 millones de URLs compartidas
  - WhatsApp: Para soportar el servicio de mensajería, con 2 millones de usuarios conectados por servidor
  - Bet365: Para el servicio de apuestas inPlay
  - ….

Erlang <-> Relación con ProLog

- ¿Qué tiene de especial Prolog que no tienen otros lenguajes que han usado antes?

```
-module(count_to_ten).
-export([count_to_ten/0]).

count_to_ten() -> do_count(0).

do_count(10) -> 10;
do_count(Value) -> do_count(Value + 1).
```

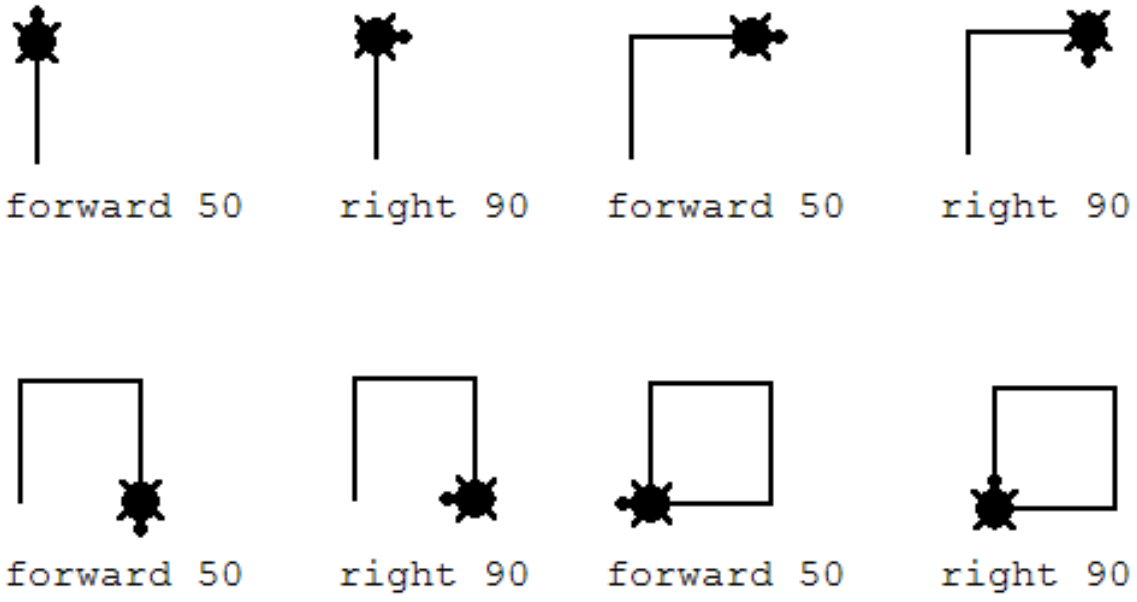
Relación con ProLog

- Hot Code Loading (Hot Swaping)

```
%% A process whose only job is to keep a counter.
%% First version
-module(counter).
-export([start/0, codeswitch/1]).

start() -> loop(0).

loop(Sum) ->
  receive
     {increment, Count} ->
        loop(Sum+Count);
     {counter, Pid} ->
        Pid ! {counter, Sum},
        loop(Sum);
     code_switch ->
        ?MODULE:codeswitch(Sum)
        % Force the use of 'codeswitch/1' from the latest MODULE version
  end.

codeswitch(Sum) -> loop(Sum).
```

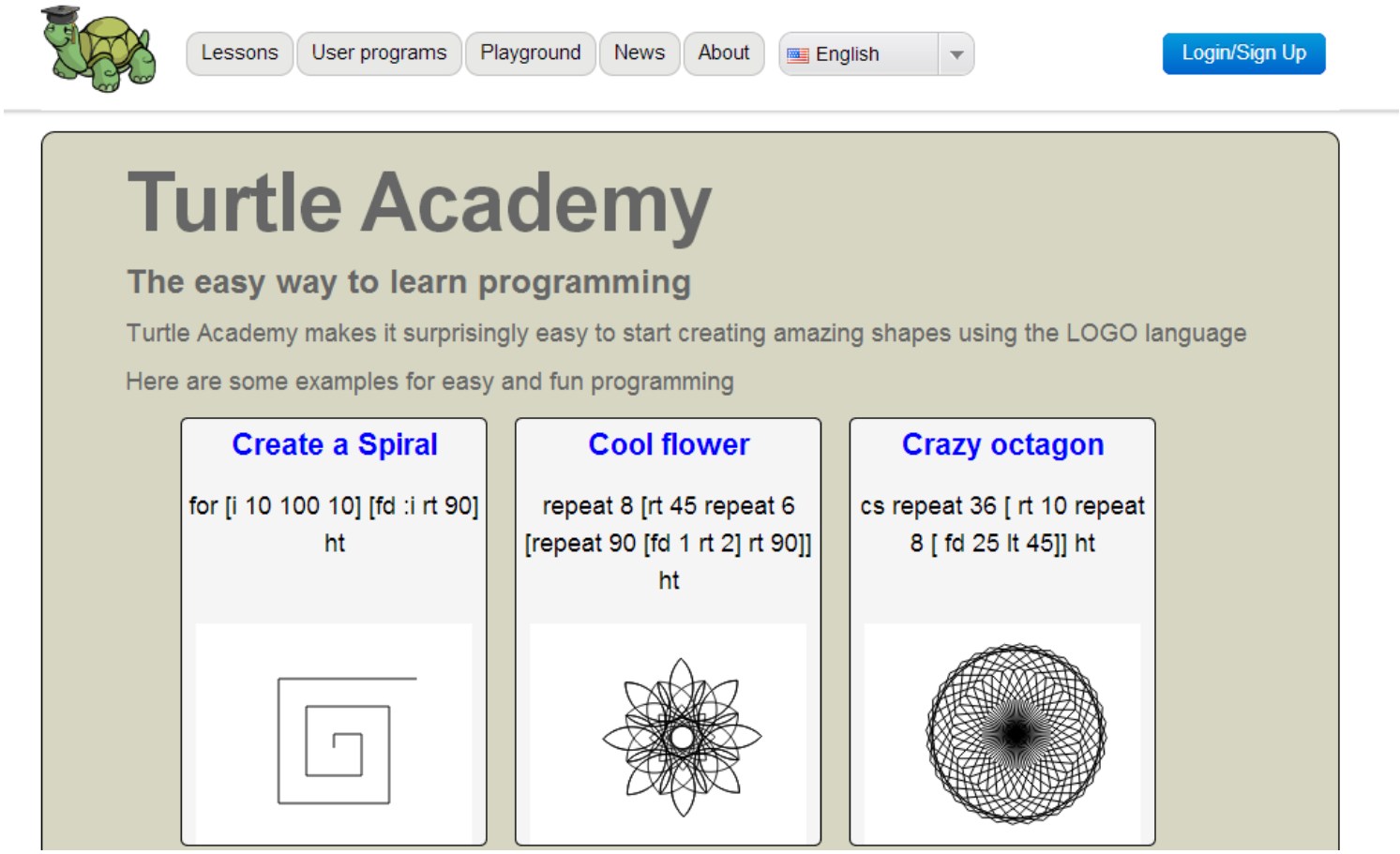
LOGO

- Creado en 1969 por Seymour Papert, con propósito pedagógico (falleció el 2016)



Por si alguien quiere probar

- Turtle academy http://turtleacademy.com/

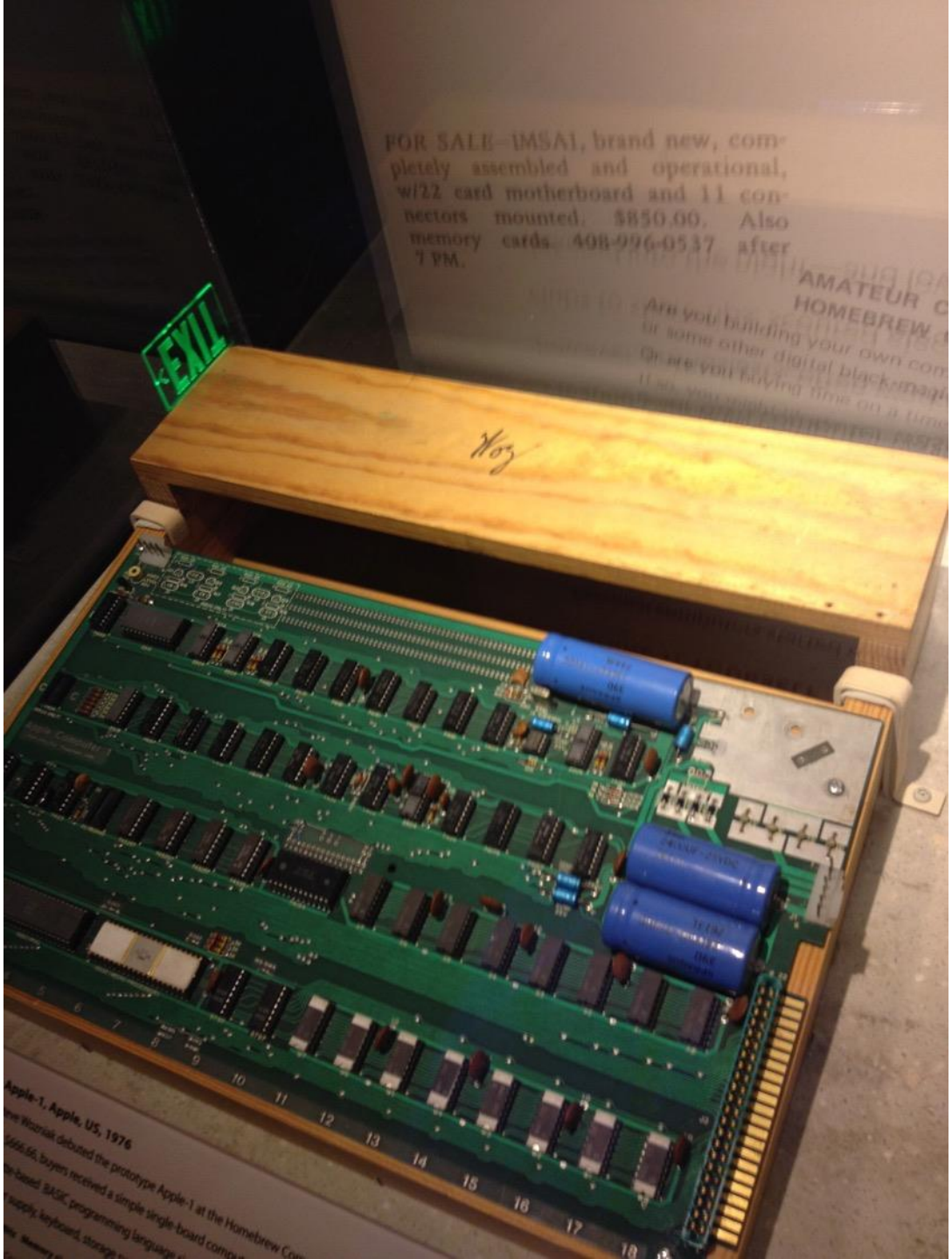


John von Neumann (left) and Manhattan Project leader J. Robert Oppenheimer, in front of Princeton's Institute for Advanced Study (IAS) computer, ca. 1952
Operational in 1952, the IAS machine was the prototype for the first generation of digital computers.

ESTABLISHING A PATTERN: VON NEUMANN AT THE IAS

The earliest advanced computers were one-of-a-kind. The machine developed by mathematician John von Neumann at the Institute for Advanced Study (IAS) was an exception.

Seventeen similar vacuum tube machines were built by governments and scientific institutions around the world, standardizing the approach of storing programs and data in a common memory.

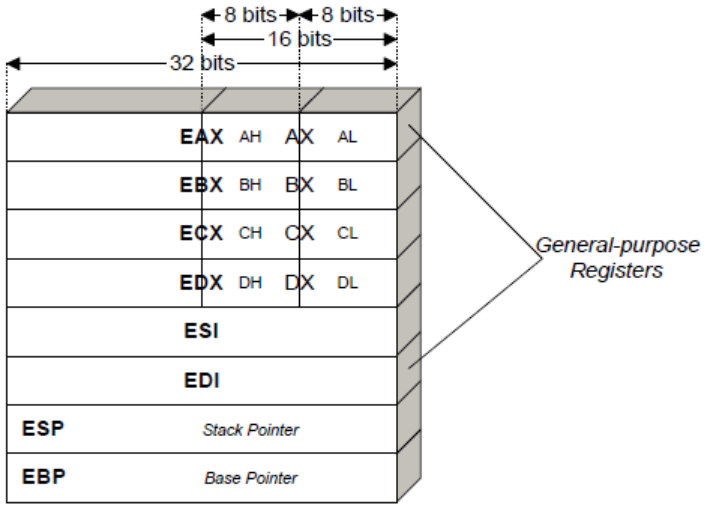

EXIT
FOR SALE—IMSAI, brand new, completely assembled and operational, w/22 card motherboard and 11 connectors mounted. $850.00. Also memory cards. 408-996-0537 after 7 PM.
AMATEUR
HOMEBREW
Apple-1, Apple, US, 1976

Cuándo fueron creados estos lenguajes?

- Assembly (195x)
- FORTRAN (1953)
- C (1969)
- Pascal (1970)
- C++ (1980)
- Java (1995)
- Javascript (1995)
- Python (1991)
- C# (2001)
- Scala (2003)

Ejemplo de Assembly x86

- **Así es como sumas dos números:**
- Poner primer numero en registro
- Poner segundo numero en registro
- Sumar los registros
- Retornar resultado




```
;n1 db 3; n2 db 7
mov eax, 3 ; podria ser mov eax, [n1]
mov ecx, 7; podria ser mov eax, [n2]
add eax, ecx
ret
```

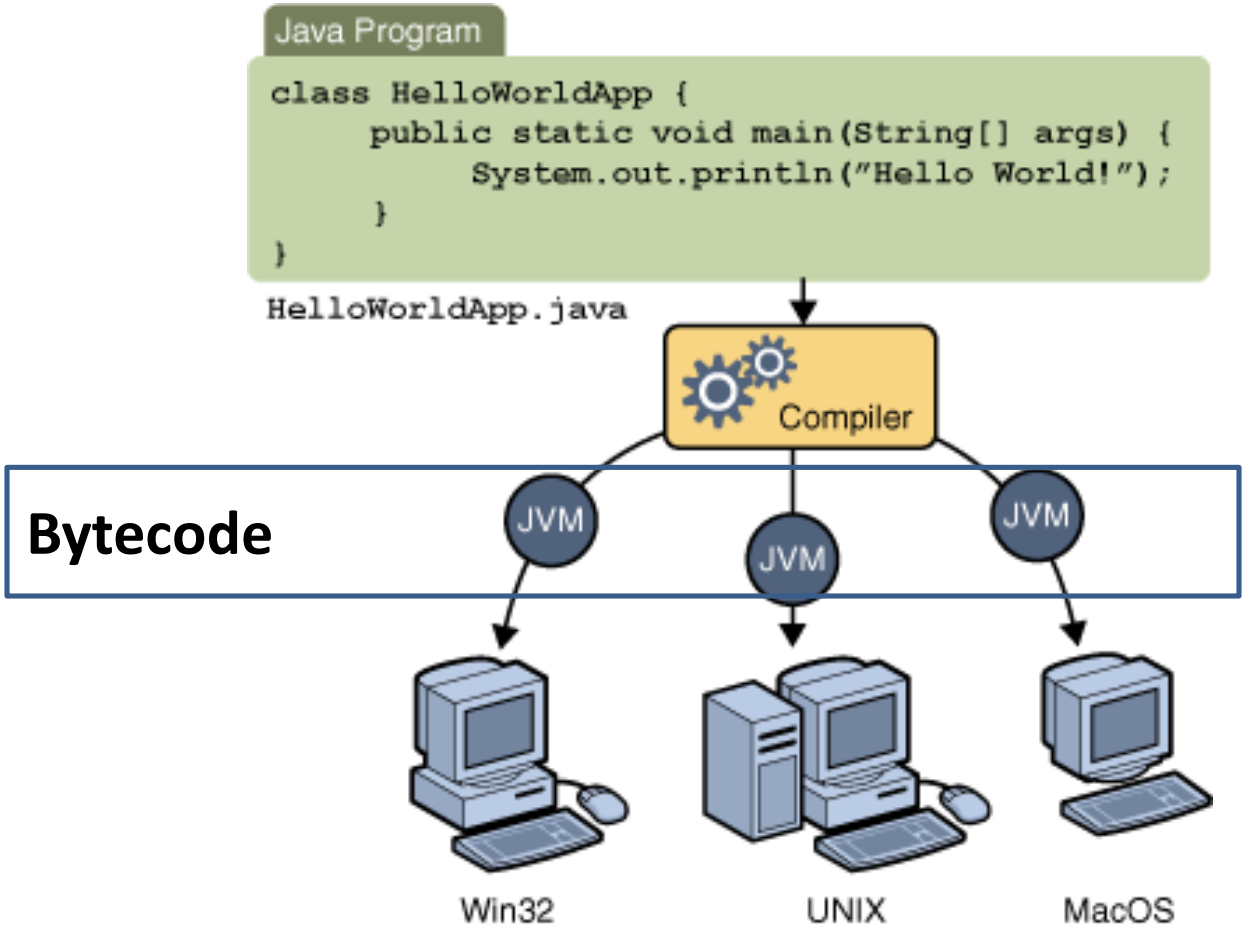
```
// equivalente en C
int a = 3;
int c = 7;
a += c;
return a;
```

Primera diferencia: Lenguaje Compilado versus Interpretado

- Compilado: FORTRAN, Pascal, C, C++
- Interpretado: Python, Ruby
- Compilado es generalmente más rapido porque apuntan directamente a la máquina/arquitectura en la cual se ejecutan.
- Interpretado tiende a ser más portable.
- Versión de lenguajes “interpretados” es más fáciles de crear porque escribir compiladores es algo difícil.

..y cómo funciona JAVA?

- Con la Java Virtual Machine



Considerando la Arquitectura: Assembly

- Es un lenguaje de bajo nivel, y no porque sea de mala calidad ;-)

- Los lenguajes “Assembly” consideran directamente la arquitectura del equipo (numero y tamaño de los registros) y por lo tanto no son portables a otras arquitecturas (como Java, por ejemplo)

Pascal: Programación estructurada

```
program Mine(output);

var i : integer;

procedure Print(var j : integer);
begin
    ...
end;

begin
    ...
    Print(i);
end.
```

C: System Language

- Lenguaje considerado como “procedural”

Procedural	Object-oriented
procedure	method
record	object
module	class
procedure call	message

POO: Hello World en C++ y Java

========== C++ ==========

```
#include <iostream>
int main() {
        //comentario
        std::cout << "Hello World!";
}
```

========== JAVA ==========

```
public class HelloWorld {       //comentario
        public static void main(String[] args) {
        System.out.println("Hello, World");
        }
}
```

Documentación: JavaDoc

- **Prueben pydoc o sphynx para python**

All Classes

Packages
cc.mallet.classify
cc.mallet.classify.constraints
cc.mallet.classify.constraints
cc.mallet.classify.evaluate
cc.mallet.classify.examples
cc.mallet.classify.tests
cc.mallet.classify.tui
cc.mallet.cluster
cc.mallet.cluster.clustering_s
cc.mallet.cluster.evaluate
cc.mallet.cluster.evaluate.tes
cc.mallet.cluster.examples
cc.mallet.cluster.iterator

cc.mallet.cluster

Classes
Clusterer
Clustering
Clusterings
GreedyAgglomerative
GreedyAgglomerativeByDensit
HillClimbingClusterer
KBestClusterer
KMeans
Record

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

cc.mallet.cluster

Class Clusterer

java.lang.Object
  └ cc.mallet.cluster.Clusterer

All Implemented Interfaces:
java.io.Serializable

Direct Known Subclasses:
KBestClusterer, KMeans

public abstract class Clusterer
extends java.lang.Object
implements java.io.Serializable

An abstract class for clustering a set of points.

Author:
Jerod Weinman weinman@cs.umass.edu

See Also:
Serialized Form

Constructor Summary

Hello World in C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hola Mundo!");
            Console.ReadKey();
        }
    }
```

C / C++ / Java

Manejo de memoria: C

```
1#include <stdlib.h> // needed for malloc and free!
2int *p_int = malloc(sizeof(*p_int));
3// use p_int
4free( p_int );
```

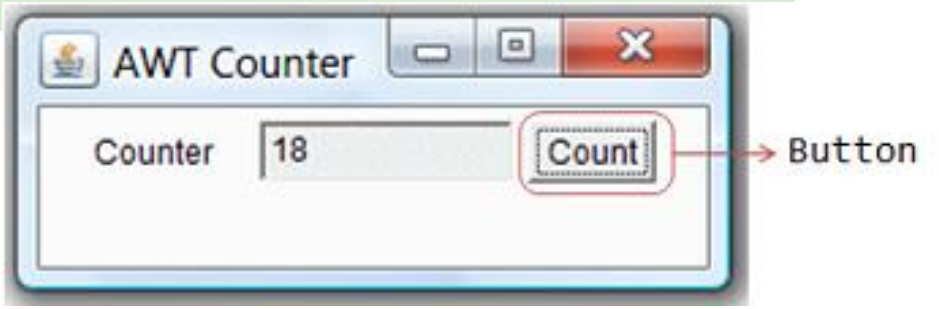
Manejo de memoria: C++

```
1int *p_int = new int;
2// use p_int
3delete p_int;
```

Manejo de memoria Java: Garbage Collector!

Java

```
import java.awt.Frame;
// Using Frame class in package java.awt
// A GUI program is written as a subclass of Frame - the top-level container
// This subclass inherits all properties from Frame, e.g., title, icon, buttons, content-pane
public class MyGUIProgram extends Frame {
// Constructor to setup the GUI components
public MyGUIProgram() {
......
}

// Other methods ...... ......
// The entry main() method
public static void main(String[] args) {
// Invoke the constructor (to setup the GUI) by allocating an instance
new MyGUIProgram();
}

}
```



PROLOG

- Logic programming language: asociado a lógica e inteligencia artifical.
- Lenguaje declarativo que expresa relaciones y permite realizar inferencias

mother_child(trude, sally).

father_child(tom, sally).

father_child(tom, erica).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).
parent_child(X, Y) :- father_child(X, Y). parent_child(X, Y) :- mother_child(X, Y).

?- sibling(sally, erica). Yes

SQL

- **Structured Query Language** orientado especialmente para DBMS relacionales
- En la clase de Information Retrieval vimos algunas procedimientos simples (bag-of-words model, TF-IDF) para buscar informacion no estructurada, pero cuando la informacion esta almacenada de forma estructurada, SQL es el estandar para consultas.

SQL – 3 ejemplos

Considerando estas tablas:

TABLA CUSTOMERS

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

TABLA ORDERS

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

SQL – EJEMPLO 1: SELECT

- SELECT [campos] FROM [tabla] WHERE [condiciones]

```
SELECT CustomerID, ContactName
FROM  Orders
WHERE Country = “Mexico”
```

Paradigma de programacion en SQL

- ¿Le dijeron en algun momento a la instrucción de SQL cómo ir a buscar la información?
- NO -> **SQL es un tipo de lenguaje declarativo**
- ¿Qué otros paradigmas de lenguajes de programación hay?
- (y por que nos interesa saber…)

Paradigmas de Programación

- Lenguajes más comunes responden a varios paradigmas

Paradigm	Description	Main characteristics	Related paradigm(s)	Critics	Examples
Imperative	Computation as statements that *directly* change a program state(data fields)	Direct assignments, common data structures,global variables		Edsger W. Dijkstra,Michael A. Jackson	C, C++,Java, PHP,Python
Structured	A style of imperative programming with more logical program structure	Structograms, indentation, either no, or limited use of, goto statements	Imperative		C, C++, Java
Procedural	Derived from structured programming, based on the concept of modular programming or the *procedure call*	Local variables, sequence, selection,iteration, and modularization	Structured, imperative		C, C++, Lisp,PHP, Python
Functional	Treats computation as the evaluation of mathematical functions avoiding state and mutable data	Lambda calculus, compositionality, formula, recursion, referential transparency, no side effects			Erlang, Haskell, Lisp, Clojure, Scala, F#

Por que sería util forzar programación funcional? rt(x) = rt(y) if x = y

```
globalValue = 0;

integer function rq(integer x)
begin
     globalValue = globalValue + 1;
     return x + globalValue;
end

integer function rt(integer x)
begin
     return x + 1;
end
```

Si escribiésemos un loop con la F(x)

- Un compilador podría detectar una función dentro de un loop y optimizar la forma de referenciarla y escribirla en código de máquina SOLAMENTE si la función no depende del estado de ejecución del programa

```
While (i < 1000)
    rq(i)
```

Depende de variable global definida en tiempo de ejecución

```
While (i < 1000)
    rt(i)
```

Este sí cumple con transparencia referencial

Otros lenguajes - LoLCode

-
-

Now with doge!

LOLCODE is an esoteric programming language inspired by the funny things that cats say on the Internet.

Learn more about the language

lci is a correct, portable, fast, and precisely documented interpreter for LOLCODE written in C.

Download source

GitHub project page

Problems? Check out the mailing list or file a bug report.

LOLCODE

Hagamos un “Hola Mundo” en LOLCODE

LOLCODE 1

```
HAI 1.2

VISIBLE "Hai world"

KTHXBYE
```

LOLCODE 2

- Declarar e inicializar una variable
- Mostrarla en Pantalla

I HAS A VARIABLE ITZ <var>

LOLCODE 3

- Agregar comentarios

BTW

LOLCODE 4

- Solicitar al usuario input desde teclado

GIMMEH

LOLCODE 5

- Una bifurcación (IF)

```
…, O RLY?
        YA RLY
                …
        NO WAI
                …
OIC
```

LOLCODE 6

- CONTADOR

IM IN YR LOOP

…

IM OUTTA YR LOOP

…otros

- Switch… case

<expression>
WTF?
  OMG <value literal>
    <code block>
 [OMG <value literal>
    <code block> …]
 [OMGWTF
    <code block>]
OIC

¿Cómo prepararme rápidamente?

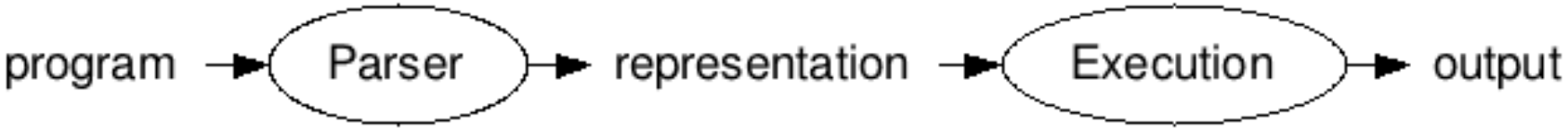
- Venir a ayudantías
- CodeCademy: Basic Web Projects
  - https://www.codecademy.com/en/tracks/projects
- W3Schools: tutorial javascript
  - https://www.w3schools.com/js/default.asp
- Libros de referencia:
  - Guia paso a paso: You Don’t Know Javascript
  - Tradicional: Javascript The Definitive Guide

¿Pero cómo empezar?

- El mejor lugar: la consola del navegador que me permite ejecutar código en javascript.
- Click con el botón derecho sobre cualquier parte de la página y seleccionar “Inspect”
  - Hola Mundo
  - alert
  - Cambiar estilos

¿Y cómo escribo mi propio lenguage?

- Necesitas un parser (sintaxis) y luego otro programa que implementa la ejecución




```
>> program = "(begin (define r 3) (* 3.141592653 (* r r)))"

>>> parse(program)
['begin', ['define', 'r', 3], ['*', 3.141592653, ['*', 'r', 'r']]]


>>> eval(parse(program))
28.274333877
```

code from (How to Write a (Lisp) Interpreter (in Python))
http://norvig.com/lispy.html

¿Y cómo escribo mi propio lenguage?

- Necesitas un parser (sintaxis) y luego otro programa que implementa la ejecución
- Debes partir definiendo tu lenguaje: símbolos, reglas… y debe ser Context-free

$$G = (\{S\}, \{a, b\}, P, S)$$

$S \rightarrow aSb$

$S \rightarrow ab$

Equivalente a $\{a^n b^n : n \geq 1\}$

Si el tiempo da

- Ver ejemplo en

http://en.wikipedia.org/wiki/Context-free_grammar

Algebraic expressions [edit]

Here is a context-free grammar for syntactically correct infix algebraic expressions in the variables x, y and z:

1. S → x
2. S → y
3. S → z
4. S → S + S
5. S → S - S
6. S → S * S
7. S → S / S
8. S → (S)

This grammar can, for example, generate the string

(x + y) * x - z * y / (x + x)

as follows:

S (the start symbol)
→ S - S (by rule 5)
→ S * S - S (by rule 6, applied to the leftmost S)
→ S * S - S / S (by rule 7, applied to the rightmost S)

Paso de ejecución

Here is the definition of `eval`. Each of the nine cases in the table above has a line or two or three here, and the definition of `eval` needs nothing but those nine cases. The `eval` function takes two arguments: an expression, `x`, and an *environment*, `env`. An environment is a mapping from variable names to their values and will be covered in depth in the next section.

```
def eval(x, env=global_env):
    "Evaluate an expression in an environment."
    if isa(x, Symbol):              # variable reference
        return env.find(x)[x]
    elif not isa(x, list):          # constant literal
        return x
    elif x[0] == 'quote':           # (quote exp)
        (_, exp) = x
        return exp
    elif x[0] == 'if':              # (if test conseq alt)
        (_, test, conseq, alt) = x
        return eval((conseq if eval(test, env) else alt), env)
    elif x[0] == 'set!':            # (set! var exp)
        (_, var, exp) = x
        env.find(var)[var] = eval(exp, env)
    elif x[0] == 'define':          # (define var exp)
        (_, var, exp) = x
        env[var] = eval(exp, env)
    elif x[0] == 'lambda':          # (lambda (var*) exp)
        (_, vars, exp) = x
        return lambda *args: eval(exp, Env(vars, args, env))
    elif x[0] == 'begin':           # (begin exp*)
        for exp in x[1:]:
            val = eval(exp, env)
        return val
    else:                           # (proc exp*)
```

Sugiero chequerar tambien: https://sites.google.com/a/bodik.org/cs164/directory/pa1

¡Gracias!