



Paradigmas de Programación (EIF-400) Introducción LP y Paradigmas Parte A

CARLOS LORÍA-SÁENZ

AGOSTO 2023

EIF/UNA

Propósito

2

- ▶ Un **modelo conceptual de estudio** de lenguajes-paradigmas-traducción
- ▶ Que sirva para estudiar y comparar variantes en los casos de estudio



Objetivos

3

- ▶ Un “vistazo” en el contexto y temas del curso
- ▶ Noción de paradigma y relación con lenguajes y su compilación
- ▶ Clases de Paradigmas y origen. Ejemplos: **FP**, **LP**, **OOP**
- ▶ Evolución de los lenguajes (tendencias, mercado, historia)
- ▶ Tour sobre algunos lenguajes representativos de sus paradigmas
- ▶ Esbozo de historia de algunos lenguajes icónicos
- ▶ Consideraciones especiales de arquitectura y diseño. (ejemplos: asincronía, concurrencia)
- ▶ Temas básicos y relevantes sobre compilación

Conceptos

4

- ▶ **Contexto**: Paradigma, Lenguaje, Programación
- ▶ **Declarativo** vs **Operativo**: Balance **Qué** versus **Cómo**
- ▶ **Compilación/Interpretación**: sintaxis/semántica, **Parsing/Typing**. Dinámico/estático. Scope/binding.
- ▶ **Abstracciones**: Máquinas, Máquinas virtuales. Compilación JIT/AOT
- ▶ **Abstracciones**: Patrones datos/control (clásicos y alternativos)
- ▶ **Patrones** (OOP-)imperativo vs (OOP)-FP-declarativos
- ▶ Primeras nociones de FP/LP
- ▶ **Evolución** paradigmas/lenguajes (**fuerzas evolutivas**)
- ▶ **Perfil histórico** de algunos lenguajes icónicos

Ejercicios

5

- ▶ Este material contiene ejercicios que corresponden a puntos de evaluación en trabajos grupales e individuales

Contexto: Paradigma, Lenguaje, Compilador



Compilador = Traductor

Contexto del curso

7

- ▶ Paradigmas/Lenguajes: ¿Por qué estudiarlos?
 - ▶ Programador polígloto
 - ▶ Competitividad profesional
 - ▶ Tendencias de mercado:
 - ▶ Especial ¡Al generativa!
- ▶ Dos sitios interesantes
 - ▶ Historia (genealogía)
 - ▶ Lista (> 2000 y sigue contando)

Lenguaje de Programación

8

Article Talk

Programming language

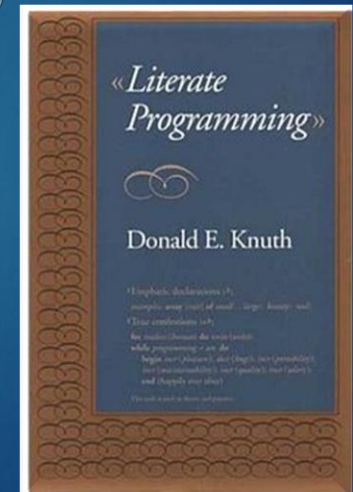
From Wikipedia, the free encyclopedia

A **programming language** is a [formal computer language](#) or [constructed language](#) designed to communicate [instructions](#) to a [machine](#), particularly a [computer](#). Programming languages can be used to create [programs](#) to control the behavior of a machine or to express [algorithms](#).

¿Programar?

9

- ▶ ¿Qué es “programar”?
- ▶ Reflexione sobre esta frase:
- ▶ D. Knuth: “*Programming is the art of telling another human being what one wants the computer to do*”
- ▶ ¿Implicación sobre un lenguaje de programación?
- ▶ ¿Y qué es en la era de la AI generativa?



Ejercicio para ChatGPT



10

- ▶ **Prompt:** *“Escriba un algoritmo que calcule el promedio de los cuadrados de una lista no vacía de números. Hágalo en tres estilos: a) imperativo b) funcional c) lógico a la Prolog. Los dos primeros en Java”*

Respuesta de ChatGPT (recortada por espacio)

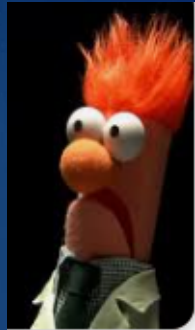
```
public static double calcularPromedioCuadrados(int[] numeros) {  
    int sumaCuadrados = 0;  
    for (int i = 0; i < numeros.length; i++) {  
        sumaCuadrados += numeros[i] * numeros[i];  
    }  
    return (double) sumaCuadrados / numeros.length;  
}
```

```
public static double calcularPromedioCuadrados(int[] numeros) {  
    return Arrays.stream(numeros)  
        .mapToDouble(num -> num * num)  
        .average()  
        .orElse(0);  
}
```

Respuesta de ChatGPT

12

```
calcular_promedio_cuadrados([], 0).  
calcular_promedio_cuadrados([H|T], Promedio) :-  
    calcular_promedio_cuadrados(T, PromedioResto),  
    Cuadrado is H * H,  
    length([H|T], Length),  
    Promedio is (Cuadrado + PromedioResto) / Length.
```



Respuestas de Bart

- ▶ Solo puede en Python (versión libre).

```
def average_squares(list):  
    sum = 0  
    for number in list:  
        sum += number ** 2  
    return sum / len(list)
```

```
def average_squares(list):  
    return sum(number ** 2 for number in list) / len(list)
```

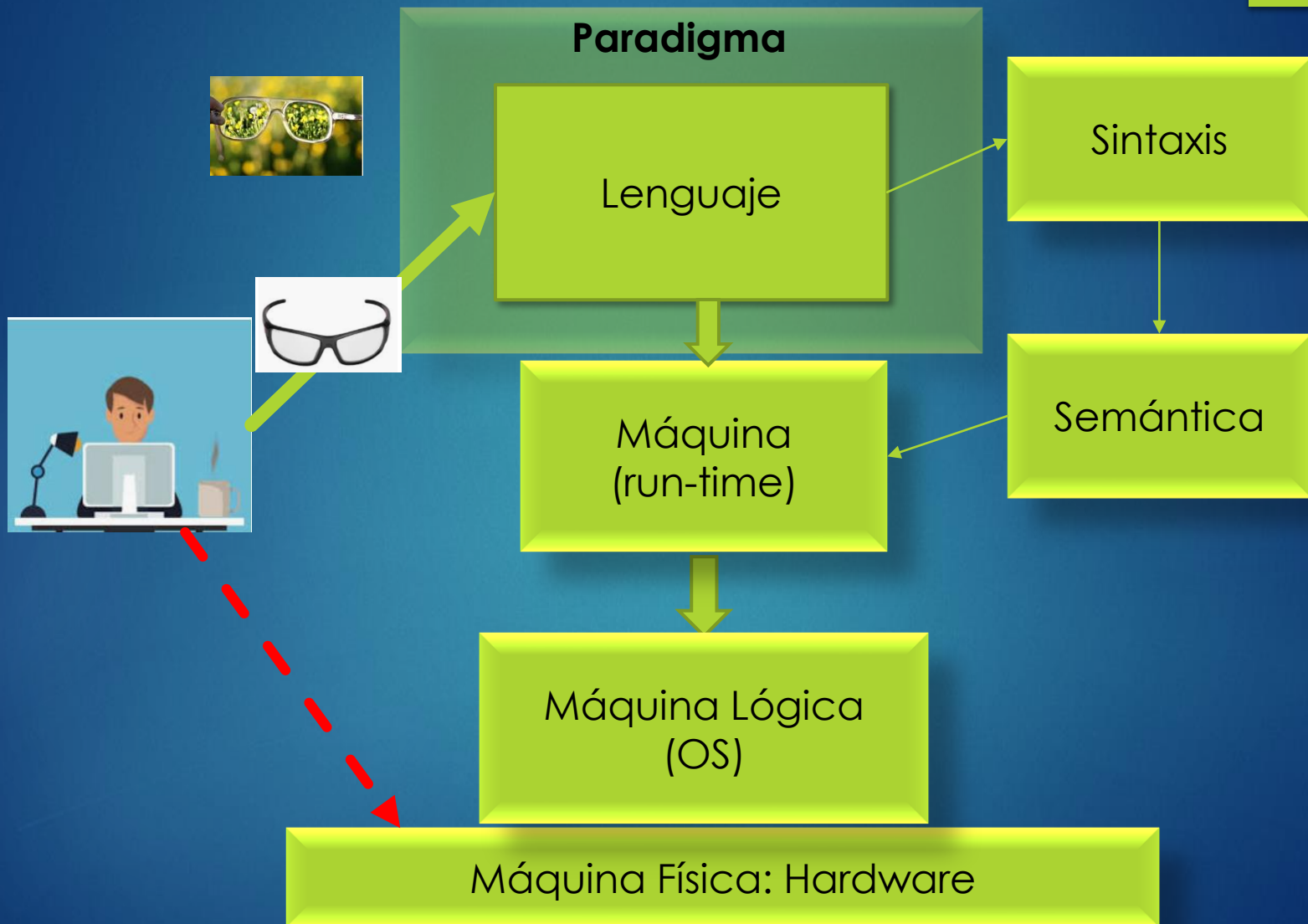
```
average_squares(L) :-  
    length(L, N),  
    sum([number ** 2 | L]) / N.
```



Big oops!

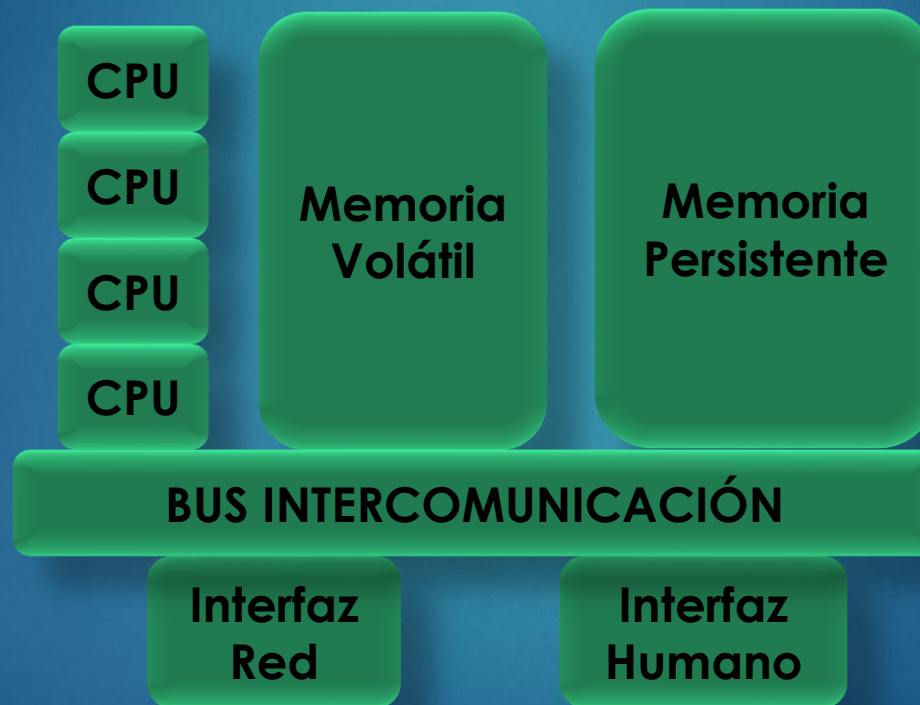
Elementos de interés

14



Máquina: Abstracción Básica

15



¿Qué es lo
ESENCIAL?

Abstracción más básica

16

- ▶ Esencial de Máquina
 - ▶ Operaciones
 - ▶ Datos
- ▶ Programa almacenado: von Neumann
- ▶ Programa = datos en la memoria que codifican operaciones
- ▶ No son “distintos” de los datos que los programas procesan
- ▶ Es posible escribir un programa que procesa programas: sistema operativo, compilador, etc.

Abstracciones matemáticas

17

- ▶ “Máquinas Operacionales”
 - ▶ Autómatas Finitos (FA)
 - ▶ Autómatas de Pila (PDA)
 - ▶ Máquinas de Turing (TM)
- ▶ “Modelos menos operacionales”
 - ▶ Cálculo λ

Contexto: Paradigmas

18

► ¿Paradigmas?

Programming paradigms are a way to classify programming languages based on their features. Languages can be classified into multiple paradigms.

Rust (programming language)

From Wikipedia, the free encyclopedia

Rust is a multi-paradigm programming language focused on performance and safety, especially safe concurrency.^{[15][16]} Rust is syntactically similar to C++,^[17] and provides memory safety without using garbage collection.

Rust was originally designed by Graydon Hoare at Mozilla Research, with contributions from Dave Herman, Brendan Eich, and others.^{[18][19]} The designers refined the language while writing the Servo layout or browser engine,^[20] and the Rust compiler. The compiler is free and open-source software dual-licensed under the MIT License and Apache License 2.0.

Rust has been the "most loved programming language" in the Stack Overflow Developer Survey every year since 2016.^[21]

Contents [hide]

- 1 Design
 - 1.1 Performance of idiomatic Rust
 - 1.2 Syntax
 - 1.3 Memory safety
 - 1.4 Memory management
 - 1.5 Ownership
 - 1.6 Types and polymorphism

Rust



Official Rust logo

Paradigms	Multi-paradigm: concurrent, functional, generic, imperative, structured
Designed by	Graydon Hoare
First appeared	July 7, 2010; 10 years ago

Otro ejemplo: Typescript

19

TypeScript

From Wikipedia, the free encyclopedia

For the typed instance analogous to a handwritten document, see [manuscript](#).

TypeScript is a [programming language](#) developed and maintained by [Microsoft](#). It is a strict syntactical [superset](#) of [JavaScript](#) and adds optional [static typing](#) to the language. TypeScript is designed for the development of large applications and [transcompiles](#) to JavaScript.^[5] As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs.

TypeScript may be used to develop JavaScript applications for both [client-side](#) and [server-side](#) execution (as with [Node.js](#) or [Deno](#)). There are multiple options available for [transcompilation](#). Either the default TypeScript Checker can be used,^[6] or the [Babel](#) compiler can be invoked to convert TypeScript to JavaScript.

TypeScript supports definition files that can contain type information of existing JavaScript libraries, much like [C++ header files](#) can describe the structure of existing [object files](#). This enables other programs to use the values defined in the files as if they were statically typed TypeScript entities. There are third-party header files for popular libraries such as [jQuery](#), [MongoDB](#), and [D3.js](#). TypeScript headers for the [Node.js](#) basic modules are also available, allowing

TypeScript



Paradigm	Multi-paradigm: functional, generic, imperative, object-oriented
Designed by	Microsoft
Developer	Microsoft
First appeared	1 October 2012; 8 years ago ^[1]

Conceptos: static typing, transcompilación (transpilación)

Paradigmas: visión “romántica”



20

- ▶ Unos “anteojos” para abstraer la realidad usando ciertos conceptos y patrones de referencia pre-establecidos
- ▶ OOP: objetos primero, propiedades y métodos (operaciones) subordinados. Herencia. Mutabilidad y control imperativo.
- ▶ FP: funciones (operaciones) primero. Composición, inmutabilidad. Poco “control imperativo”
- ▶ LP: relaciones primero, operaciones lógicas, inmutabilidad. Poco “control imperativo”
- ▶ MP: multi-paradigma. Muestra diferentes facetas paradigmáticas

Contexto: Ejercicio

21

- ▶ Preferiblemente en grupos de 4 oficiales
- ▶ Busque en Wikipedia u otra fuente el paradigma de cada uno. Haga una matriz comparativa

- | | |
|-----------|--------------|
| ▶ Fortran | ▶ Python |
| ▶ Scheme | ▶ Prolog |
| ▶ Cobol | ▶ Javascript |
| ▶ C | ▶ Typescript |
| ▶ C++ | ▶ Go |
| ▶ Java | ▶ Rust |
| ▶ C# | ▶ Swift |
| ▶ Erlang | ▶ Scala |
| | ▶ Kotlin |
| | ▶ R |
| | ▶ F# |

Contexto: “Antes” era sólo programar...

- ▶ Paradigmas y lenguajes
- ▶ Desarrollar es mucho más que programar
- ▶ Mucho más que “lenguajes” core
- ▶ Antes era si goto o while, si struct o class
- ▶ Ahora: IDE, Libs/APIs, frameworks, patrones, arquitecturas,....
- ▶ ¿Qué cambió?



Contexto Paradigmas: la historia “corta”

- ▶ Assembler → Alto Nivel (goto) → Programación Estructurada, FP, LP → OOP → F(R)OOP
- ▶ Archivos planos → Indexados → Redes → SQL → NSQL
- ▶ Editores Texto → IDEs
- ▶ Interacción consola → Ambientes gráficos
- ▶ Lenguajes → bibliotecas → frameworks (patrones, arquitecturas)
- ▶ Decenas de programadores (élite) → decenas de miles (masiva)
- ▶ Programación individual → Grupal (local) → Masiva Distribuida (global, ubicua)
- ▶ Código Privado → Comunidades → Open Source/Estándares
- ▶ Desarrollo ad hoc → waterfall → prototipos → ágil
- ▶ Deployment diario, ciclos cortos (CD-CI) (dev-ops)
- ▶ Mainframes → PC → Redes locales → Redes Globales → Móvil → Cloud
- ▶ Mono CPU → Multicore (KB → TB; Hz → GHz)
- ▶ Cloud computing
- ▶ App transaccional → IA/IOT
- ▶ Academia/Gobierno/Empresa → Hogar → individuo



Popularidad de Lenguajes

24

- ▶ Herramientas para “*medir indirectamente*” la popularidad de lenguajes y herramientas
- ▶ Representan tendencias.
- ▶ No todas son exactamente “estadísticas formales” pero son un parámetro interesante
- ▶ Ejemplos
 - ▶ Encuesta stackoverflow (formal en la comunidad SO)
 - ▶ [Tiobe](#) (basado en sitios)
 - ▶ [PyPL](#) (basada en búsquedas de tutoriales)
 - ▶ [Github pull request](#) (basada en proyectos open source)

Contexto: Encuesta stackoverflow 2023

25



2023
Developer
Survey

In May 2023 over 90,000 developers responded to our annual survey about how they learn and level up, which tools they're using, and which ones they want.

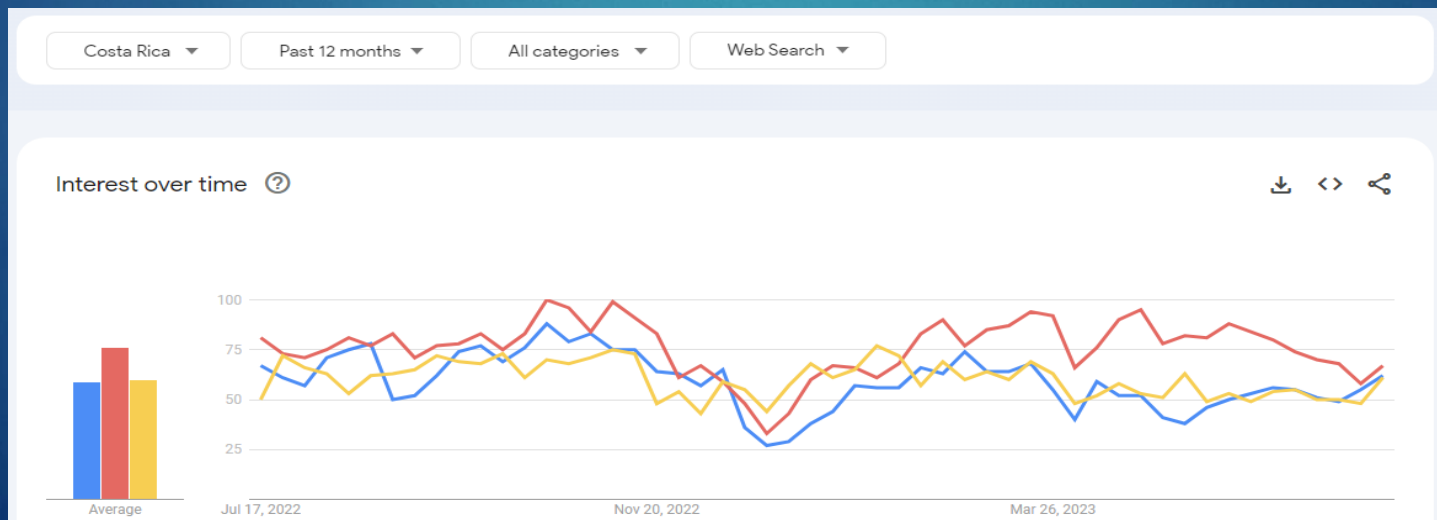
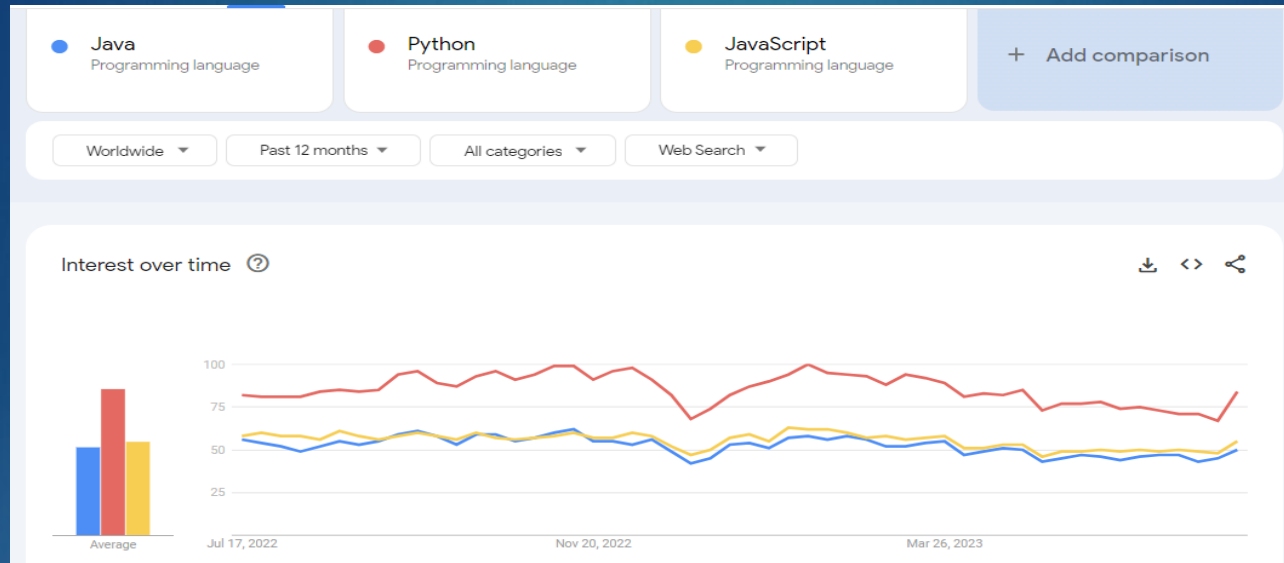


Technology

AI

Contexto: Google trends

26




Contexto: Ejercicio

27

- ▶ En grupos de 4, preferiblemente los oficiales. Fecha y forma de entrega será comunicada en clase
- ▶ Consiga 10 ofertas de empleo para puestos en ingeniería informática en Costa Rica. Debe ser puesto que implique desarrollar.
- ▶ Extraiga y sintetice datos sobre los requisitos (saber, competencia) en lenguajes y herramientas asociadas que se piden para cada puesto
- ▶ Determine cuantitativa y cualitativamente si los miembros del grupo tienen por medio de los cursos llevados o por llevar y en qué medida acceso a esos saberes y competencias

Contexto: TIOBE 2023-2022

28













[About us](#)
[News](#)
[Coding Standards](#)
[TIOBE Index](#)
[Contact](#)

[Products](#)
[Quality Models](#)
[Markets](#)

Schedule a demo

when starting to build a new software system. The definition of the TIOBE index can be found [here](#).

Jul 2023	Jul 2022	Change	Programming Language		Ratings	Change
1	1			Python	13.42%	-0.01%
2	2			C	11.56%	-1.57%
3	4	▲		C++	10.80%	+0.79%
4	3	▼		Java	10.50%	-1.09%
5	5			C#	6.87%	+1.21%
6	7	▲		JavaScript	3.11%	+1.34%
7	6	▼		Visual Basic	2.90%	-2.07%
8	9	▲		SQL	1.48%	-0.16%
9	11	▲		PHP	1.41%	+0.21%
10	20	▲▲		MATLAB	1.26%	+0.53%

PYPL Popularity of Programming Language

Worldwide, Jul 2023 :

Rank	Change	Language	Share	1-year trend
1		Python	27.43 %	-0.2 %
2		Java	16.19 %	-1.0 %
3		JavaScript	9.4 %	-0.1 %
4		C#	6.77 %	-0.3 %
5		C/C++	6.44 %	+0.2 %
6		PHP	5.03 %	-0.4 %
7		R	4.45 %	+0.1 %
8		TypeScript	3.02 %	+0.3 %
9	↑	Swift	2.42 %	+0.4 %
10	↑↑↑↑	Rust	2.15 %	+0.6 %
11	↓↓↓	Objective-C	2.13 %	+0.0 %
12	↓	Go	2.01 %	+0.0 %

Programming Language
yzing how often language
n Google.

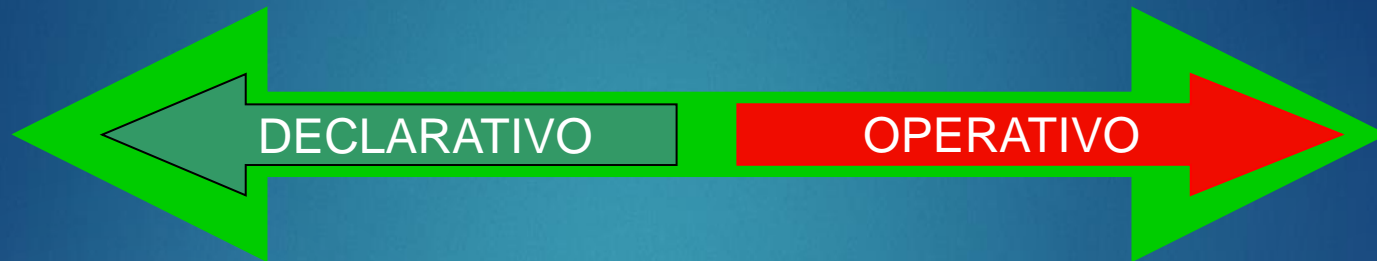
rch, the more popular the language
icator. The raw data comes from

Escala semántica

30

¿Qué?

¿Cómo?



Faceta Declarativa

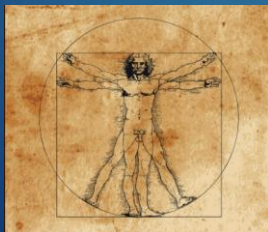
Faceta Operacional

Lenguaje Natural

Lenguaje matemático

Lenguaje de dominio

Ser Humano



Previsible:
Disminución del rol
del ser humano en
roles bien
definidos (2023)

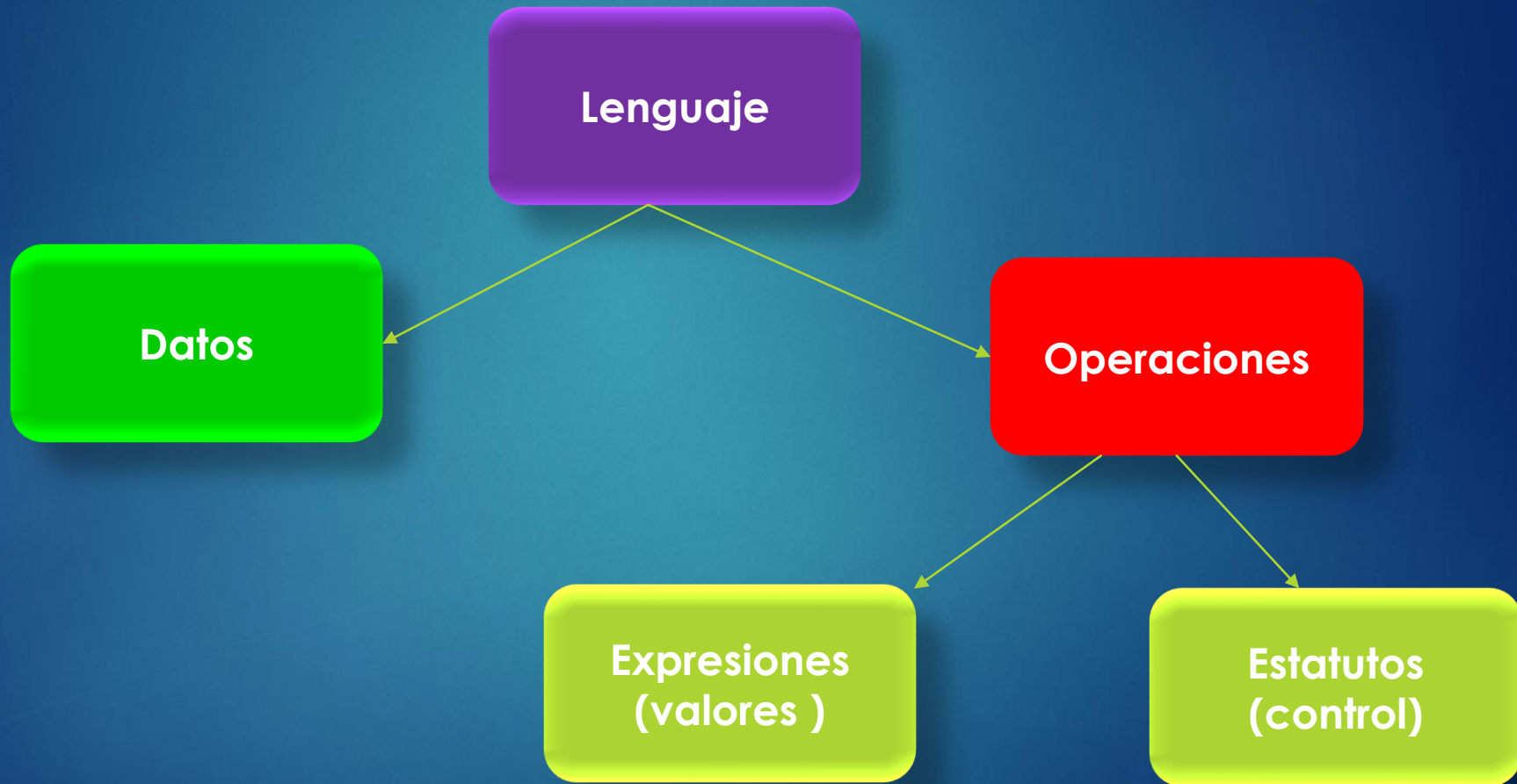
Lenguaje computacional

Agente Computacional



Lenguaje = Datos + Operaciones

31



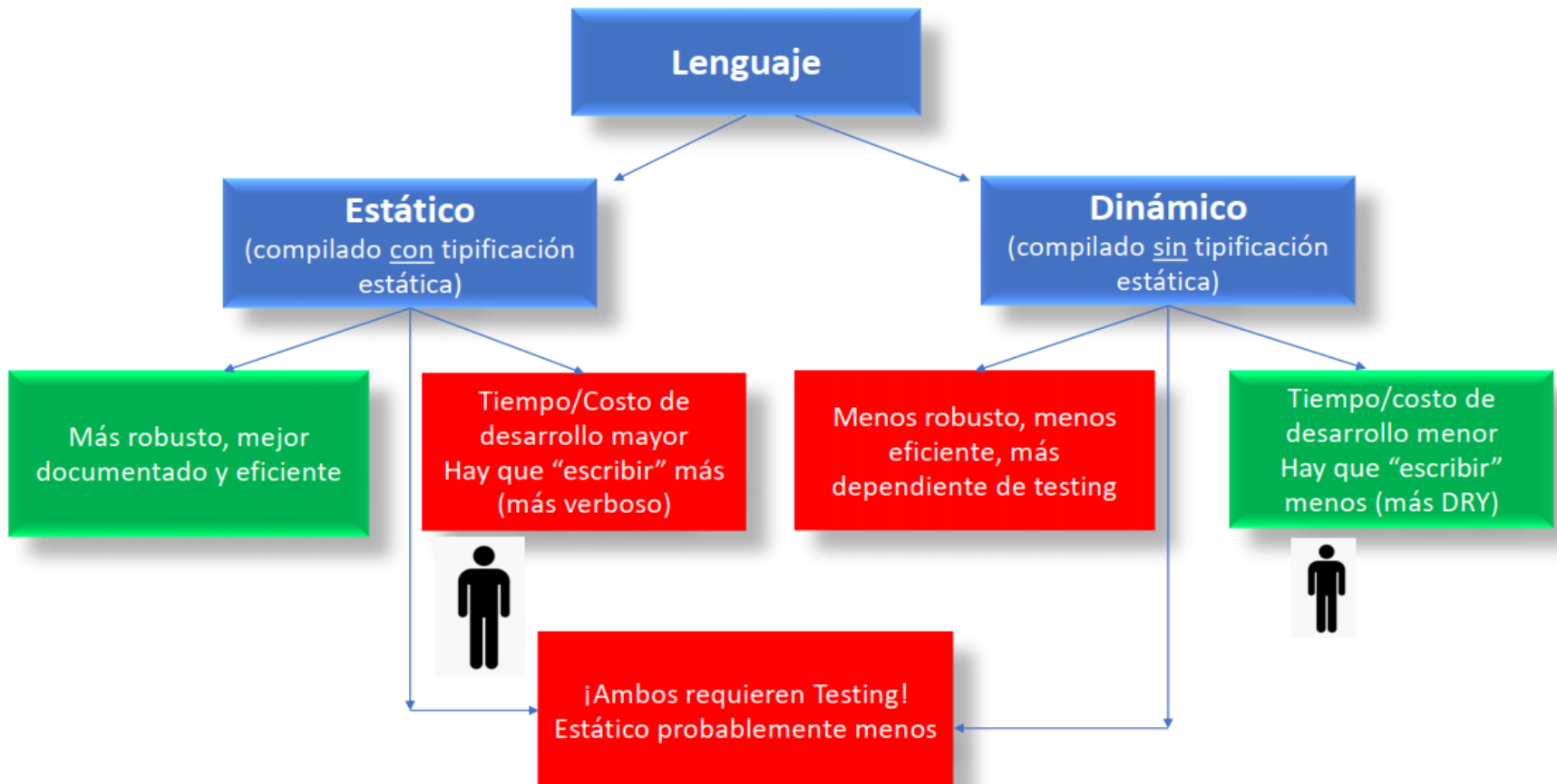
Dinámico versus Estático

32

- ▶ **Lenguajes compilados**: dos tiempos
 - ▶ **Tiempo compilación** pesado (tipificación estática “pesada”)
 - ▶ **Tiempo ejecución** (tipificación dinámica “liviana”). Sólo existe si hubo compilación
- ▶ **Lenguajes interpretados** (dinámicos)
 - ▶ Tiempo “compilación” débil o hasta inexistente
 - ▶ Tiempo de ejecución con tipificación dinámica “fuerte” durante la corrida

Estático versus Dinámico

33



Ejercicio: Conceptos lenguajes

- ▶ En grupos, preferiblemente los oficiales
- ▶ Investigar y presentar reporte sobre:
 - ▶ Sistema de tipos
 - ▶ Lenguaje fuertemente-tipado,
 - ▶ Lenguaje débilmente tipado,
 - ▶ “Duck-typing”,
 - ▶ Generics (polimorfismo paramétrico)
 - ▶ Inferencia de tipos
 - ▶ Tipos nominales
 - ▶ Tipos estructurales

Capas de abstracción

35



Ejemplos Abstracción

36



Evolución Escala Semántica

37



Lenguajes: áreas y propósitos

- ▶ Propósito **general/específico**: usuario
- ▶ Aplicaciones empresariales (usuario final)
 - ▶ Web, móvil
 - ▶ Office automation
 - ▶ Data Mining/Machine Learning/Data Science
- ▶ Programación de sistemas (developer)
 - ▶ Sistemas operativos (incluye empotrados)
 - ▶ Servidores (Web, DB)
 - ▶ IDE, tools, SDKs
 - ▶ Frameworks
 - ▶ Juegos

¿"Poder" de un Lenguaje de Programación?

- ▶ Reflexione sobre esta pregunta: ¿Es CSS un lenguaje de programación?
- ▶ ¿A qué paradigma se adhiere?
- ▶ ¿HTML?
- ▶ ¿SQL?
- ▶ ¿Cuál es la parte operacional de SQL?

Ejercicio: Turing Completo

40

- ▶ En grupos preferiblemente los oficiales
- ▶ Concepto a investigar: Turing completo
- ▶ ¿Es SQL Turing completo?

Algunos Elementos Relevantes sobre VM (RT)

- ▶ “Máquinas virtuales y frameworks” (multiplataforma vs nativo)
- ▶ JVM y CLR (.Net) (memoria manejada, GC)
- ▶ ART (Android)
- ▶ Compilación: JIT vs AOT (Java \geq 9)
- ▶ “*Write Once Run Anywhere*” (WORA)
- ▶ Los browsers: son la VM de JS de cliente
- ▶ Web basado en “lenguajes script” ¿Por qué no también el server? (Node.js)

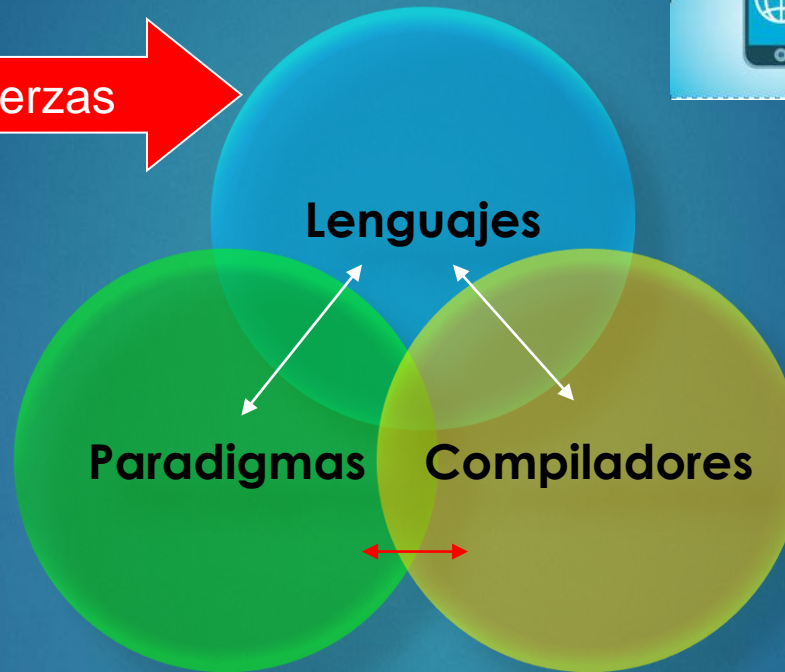
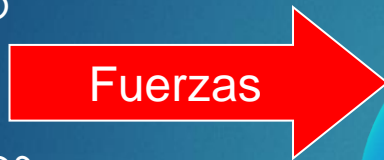
Fuerzas de evolución

42

Avance
tecnológico

Necesidades
(mercado: AI/ML
bigdata IoT)

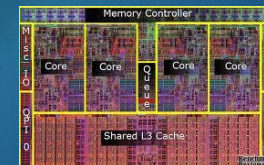
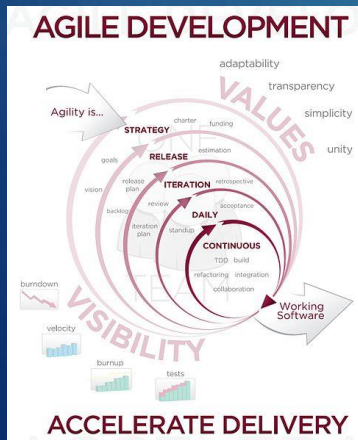
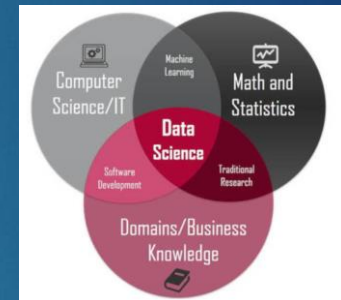
Desarrollo
veloz y ágil



Web/ Móvil



Data Science



Concurrencia
(multicore, GPU)

Virtual machines, Docker

¿Y cloud computing?

43

- ▶ Nube == Sistema Operativo/Plataforma
- ▶ Serverless, Function-as-service (FaaS)
- ▶ Ejemplos: AWS Lambda, Google Cloud Functions, Azure Functions
- ▶ Realización arquitecturas/plataformas/servicios más que con lenguajes
- ▶ No ha “permeado” tanto al nivel de lenguaje/paradigma
- ▶ Puede implicar SO, Almacenamiento (infraestructura) , Contenedores de aplicaciones

Historia Lenguaje (algunos)

44

- Fortran (54) (Primer lenguaje compilado)
- Lisp (58) (Primer FP; AI)
- Cobol (59) (estandarizado)
- Simula (62) (origen OOP)
- Basic (64) (pre-PC)
- Logo (67)
- Algol(68) (programación Estructurada, definición formal)
- Pascal (70) (un ALGOL simple)
- Awk (70)
- Scheme (70)
- C (72) (base de UNIX, sintaxis icónica)
- Smalltalk (72) (OOP)
- Prolog (72) (LP; IA)
- ML (73)
- Modula-2 (78)
- Ada (80) (DoD)
- SQL (86)

Historia (cont.)

45

- ADA (83) (diseño-por-contrato)
- C++ (83)
- Eiffel (85)
- Objective-C (86)
- Object-Pascal, Delphi (86, 95)
- Perl (87)
- Erlang(89, distribuido)
- Haskell (90)
- VisualBasic (91)
- Python (91)
- Java (91)
- Lua (93) (Brasileño!)
- R (93) (Data science)
- JavaScript (95)
- PHP (95)
- Ruby (95) (japonés)
- C# (2000) (El Java de MS)

Historia más reciente

46

- ▶ **Scala** (2003) (OOP y FP integrados en JVM). Mejor Java
- ▶ **Groovy** (2003) (Un Java-Python)
- ▶ **F#** (2005) (OCaml corriendo en .Net)
- ▶ **Clojure** (2007) (ako “Scheme” en JVM)
- ▶ **Node.js** (2009) (JS en server, V8) (impulsa ES6, ES7, ...)
- ▶ **Go** (2009) (google, concurrencia) ¿Algo entre C y Java?
- ▶ **Typescript** (MS, 2012, competidor de Dart). Pareja de Angular
- ▶ **Rust** (2010) (¿Sustituto de C++?) Primer release mayo-2015
- ▶ **Kotlin** (2011) Un mejor Java/Scala (oficial Android language)
- ▶ **Elixir** (2011) (Un Erlang renovado)
- ▶ **Hack** (2014) (Facebook, evolución de PHP)
- ▶ **Swift** (2014, Apple, MacOS, iOS; reemplazo de Objective-C)

Uno nuevo: Zig

47



[Download](#) [Learn](#) [News](#) [Source Code](#) [Join a Community](#) [♥ Sponsor the Zig Software Foundation](#)

Zig is a general-purpose programming language and toolchain for maintaining **robust**, **optimal** and **reusable** software.

GET STARTED

Latest Release: **0.9.1**

[Documentation](#)

[Changes](#)

⚡ A Simple Language

Focus on debugging your application rather than debugging your programming language knowledge.

- No hidden control flow.
- No hidden memory allocations.
- No preprocessor, no macros.

⚡ Comptime

A fresh approach to metaprogramming based on compile-time

```
const std = @import("std");
const json = std.json;
const payload =
    \\{
    \\    "vals": {
    \\        "testing": 1,
    \\        "production": 42
    \\    },
    \\    "uptime": 9999
    \\}
;
const Config = struct {
```

[Presentación en InfoWorld 2023](#)

Uno Nuevo: MOJO

Mojo  — a new
programming language
for all AI developers.

Mojo combines the usability of Python with the performance of C, unlocking unparalleled programmability of AI hardware and extensibility of AI models.