



LENGUAJES DE PROGRAMACIÓN

2020 — 2º SEMESTRE

CLASE 1:

- ▶ Información general del curso
- ▶ Motivación y contenido del curso

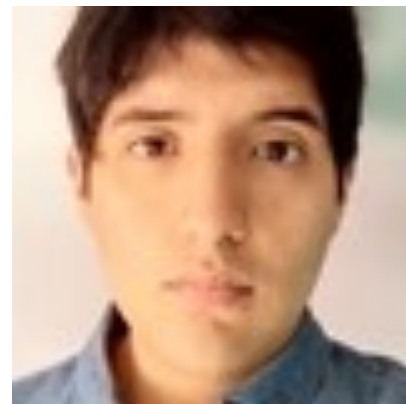
Federico Olmedo

► Información general del curso



CÁTEDRA:

FEDERICO OLMEDO



AUXILIAR/AYUDANTE:

BRYAN ORTIZ



AUXILIAR:

TOMÁS VALLEJOS



AYUDANTE:

VICENTE ILLANES

Horarios y calendario académico

CLASES DE CÁTEDRA:



Martes y Jueves, 10:15am



Zoom

CLASES AUXILIARES:



Viernes, 2:30pm



Zoom

Evaluación

■ 4 tareas

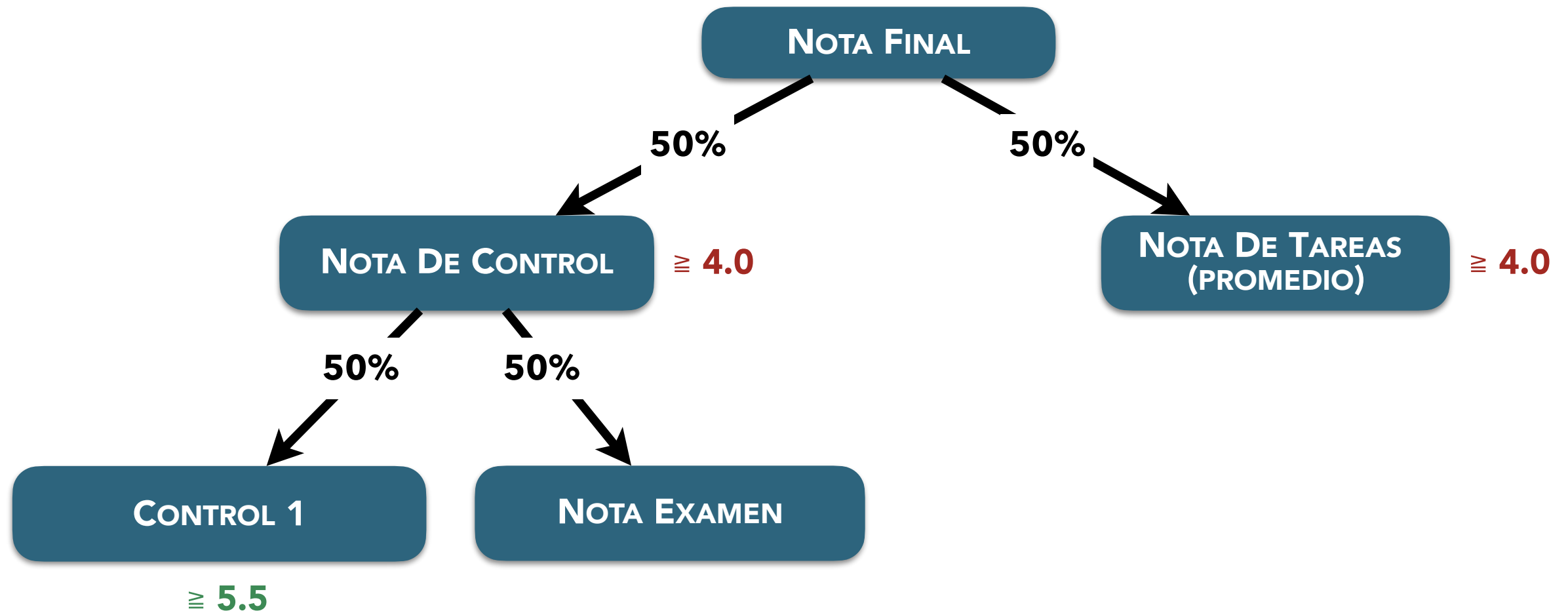
- ▶ Individuales
- ▶ Plazo de entrega: 2 semanas
- ▶ 6 días de gracia (sin penalización) a repartir entre todas las tareas.
Por tarea se permite un máximo de 2 días de gracia.
- ▶ No hay tarea recuperativa

■ 1 control

- ▶ Síncrono
- ▶ A “libro abierto”
- ▶ Semana 13

■ Examen

No se permite convalidar tareas o controles de un semestre a otro.



APROBACIÓN DEL CURSO:

Nota de control y nota de tareas ≥ 4.0 c/u

EXIMICIÓN DEL EXAMEN:

Control 1 ≥ 5.5

Plagiarismo

Toda entrega deber ser enteramente fruto del trabajo individual o de los miembros del grupo y no puede ser derivada del trabajo de otros, ya sea de fuentes publicadas como no publicadas, la web, otro estudiante, libros, materia de otros cursos (incluyendo semestres anteriores de este curso), o cualquier otra persona o programa. Se prohíbe copiar, examinar, o alterar la tarea de otra persona.

Toda violación se reportará a las autoridades de la Facultad, solicitando un sumario, poniendo en riesgo su continuidad en esta Casa de Estudios.

Whiteboard policy

Para facilitar el aprendizaje cooperativo, se permite conversar de una tarea con otros estudiantes, siempre y cuando se respete la siguiente política ([whiteboard policy](#)):

Una conversación puede tener lugar en una pizarra (o sobre papel, etc.), y debe cumplir las siguientes reglas:

- Nadie tiene permiso de tomar notas, grabar la conversación, copiar o fotografiar lo que esté escrito en la pizarra. La pizarra debe borrarse después de la discusión.
- Se debe respetar un lapso de cuatro horas después de cualquier conversación antes de empezar a trabajar en la tarea.

El hecho de que pueda recrear la solución de memoria se considera como prueba de que se entendió efectivamente.

En caso de conversar sobre la tarea con otro estudiante, todos los involucrados deben explicitarlo en la tarea, declarando con quién conversaron y sobre qué ejercicio.

PÁGINA WEB DEL CURSO:

- <https://pleiad.cl/teaching/cc4101>

MATERIAL DE REFERENCIA:

- Diapositivas y grabaciones de las clases
- [PLAI](#): Programming Languages: Application and Interpretation (1st Edition), Shriram Krishnamurthi [Capítulos 1-14, 35-37]
- [PrePLAI](#): introducción a la programación funcional en Racket
- [A Note on Dynamic Scope](#): para complementar el PLAI respecto a alcance dinámico
- [A Note on Recursion](#): para complementar el PLAI respecto a recursión
- [Learn You a Haskell](#): para complementar el PLAI respecto a Haskell
- [OOPLAI](#): apunte para la unidad de objetos

Para el material de referencia específico de cada clase consultar las diapositivas



- ▶ Anuncio noticias
- ▶ Foro consultas
- ▶ Publicación notas
- ▶ Entrega tareas
- ▶ Diapositivas
- ▶ Email profesores
- ▶

- ▶ Motivación y contenido del curso

Discussion: Programming Languages (PLs)

Your experience:

- What PLs have you used?
- Which PLs' features do you like/dislike? Why?

More generally:

- What is a PL?
- Why are new PLs created? Why are there so many?

Why are there so many PLs?

In theory

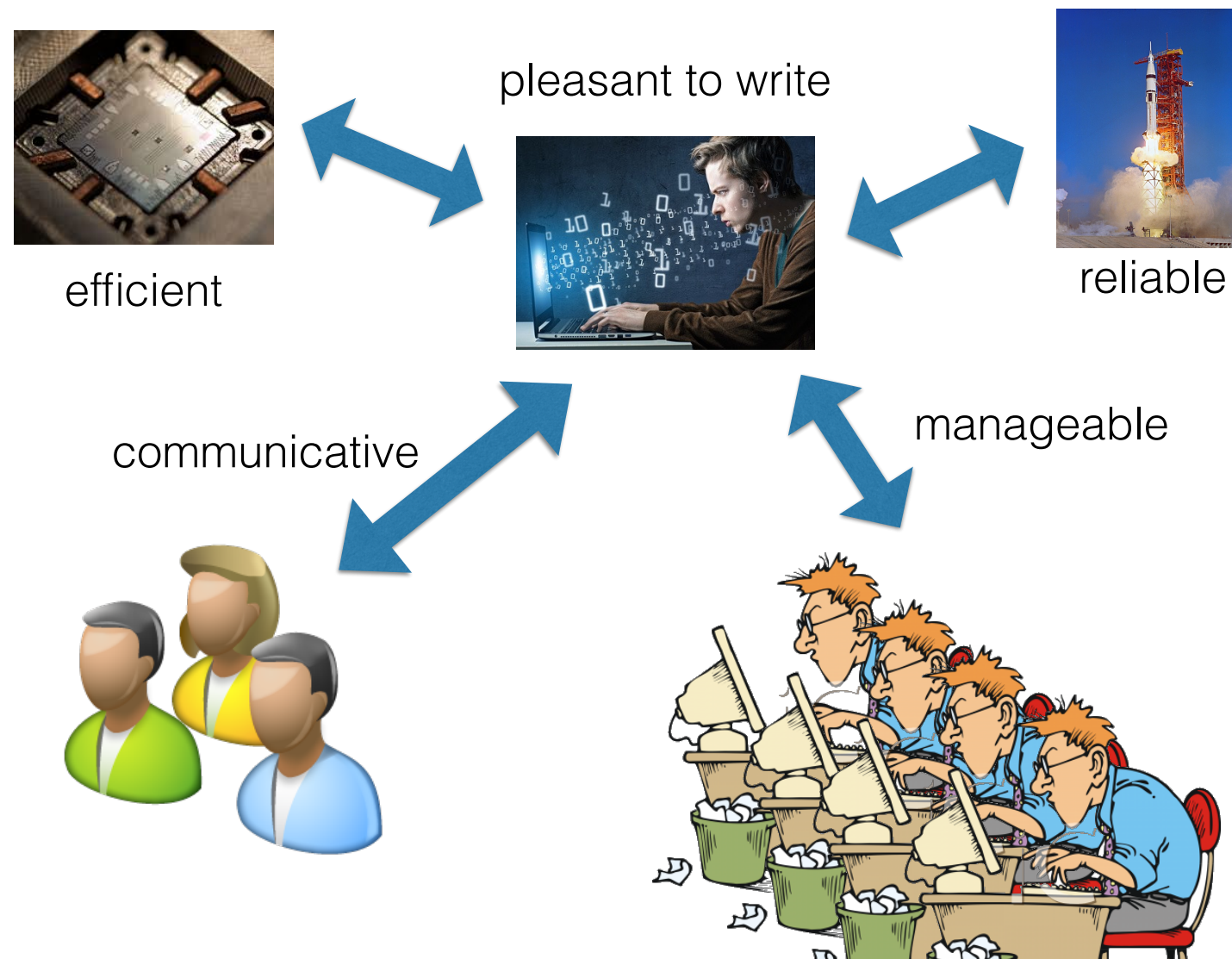
All languages are equivalent:

- They are all *Turing-complete*, i.e.
- they can compute the same things

Why are there so many PLs?

Pragmatically ...

Expressive power is not all that matters:



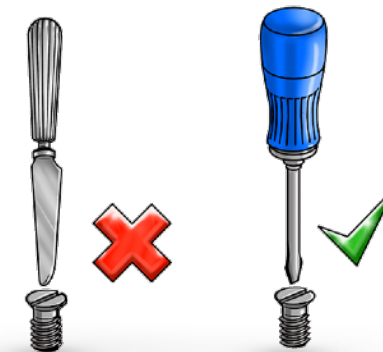
Why to study PLs?

Different problems = different languages

- ▶ Facebook or web browser in assembly?

Know your tools!!!

- ▶ Choose the right one for the right task
- ▶ Use tools effectively



What defines a PL?

Syntax: Which sequences of symbols define a valid program?
(**form** of programs)

Semantics: How does a program behave?

- ▶ What actions does it perform?
- ▶ What values does it produce?

(**meaning** of programs)

Libraries: Support for commonly used algorithms, data structures, I/O and other syscalls.

Does not tell much about the program behaviour

Here are three ways of expressing "**retrieve the *i*-th element of array *x***" in different programming languages:

- | | | |
|-----|-------------------------------|----------|
| (a) | <code>x[i]</code> | [C] |
| (b) | <code>(vector-ref x i)</code> | [Scheme] |
| (c) | <code>x[i]</code> | [Java] |

Even though they look similar, these expressions means different things:

- (a) if *i* in bounds of *x* then *x*[*i*]
 else who knows?
- (b) if *x* is not a vector then ERROR
 else if *i* is not an integer then ERROR
 else if *i* is not in bounds of *x* then ERROR
 else *x*[*i*]
- (c) if *i* is in bounds of *x* then *x*[*i*]
 else ERROR

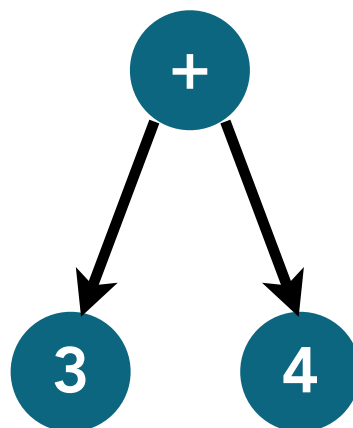
Don't get too emotional about the syntax

3 + 4 — infix notation

(+ 3 4) — prefix notation

(3 4 +) — postfix notation

All this means the same: the idealized action of adding the idealized numbers (represented by) "3" and "4".



Libraries of a PL

- Very important to programmers (and the language adoption)
- But not so relevant for the language study

Just semantics. That's all there is.

— Shriram Krishnamurthi

How shall we describe the semantics of a PL?

Natural language is not well-suited (too imprecise)!!!

Mathematics-based approaches:

- Denotational: Attaches a mathematical meaning (**element within** an appropriate **semantic domain**) to each program.
- Axiomatic: Establishes a logical relationship between the **properties that hold before and after the program execution**.
- Operational: Interprets programs as a **sequence of computational steps** (ie transitions in some sort of transition system).

"In-between" approach:

- Interpreter: The interpreter **executes the program (step-by-step)** and returns its output.

Interpreter semantics

To explain a language, write an interpreter for it!!!


- Writing forces understanding (like mathematics)
- Once written, the interpreter can be executed
- Allows for incremental modifications

But an interpreter is a program, written in a language...

- Use a simple and well-understood language,
- with already built mathematical foundations

- Dialect of Lisp, developed in 1975-1980 (MIT AI Lab)
- Functional programming language (based on the lambda calculus)
- Minimalist design: small core + powerful extension tools (macros)

Racket: A super Scheme

 **Racket** solve problems · make languages

[DOCS](#)[PACKAGES](#)[DOWNLOAD](#)

Batteries included

Cross-platform

<http://racket-lang.org/>

Powerful macros & languages

Mature, stable, open source

DrRacket IDE & tons of documentation

The best of Scheme and Lisp

- Semantics is the most important component of a PL.
- Understanding the different features/mechanisms of PLs together with their semantics is fundamental for using them effectively.

Bibliography

- Programming Languages: Application and Interpretation (1st Edition)
Shriram Krishnamurthi [\[Download\]](#)
Chapter 1.1-1.2