

CLOSURE IN ACADEMIA

Teaching Functional Programming to Students Who
Aren't Looking For It

Jim E. Newton

EPITA Research Laboratory



September 30–October 2, 2026



Companion article — scan to
download



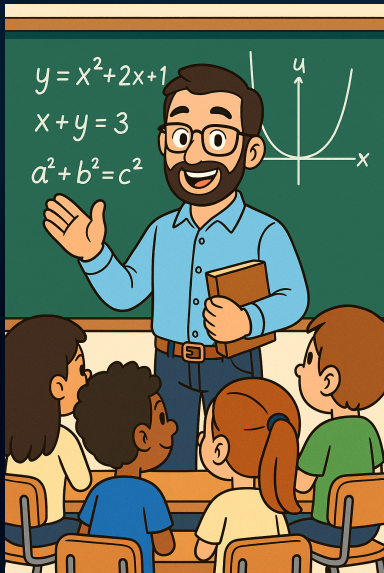
CLOSURE/
-> CONJ2026
Charlotte, NC

Overview

- The Course
- The Survey
- What Fails to Land
- What Lands
- The Grade Puzzle
- Academic Integrity
- Final Thoughts



Who am I?



Research Professor at

EPITA School of Engineering and Computer Science
Paris, FRANCE

40 years programming, mostly in lisp: emacs-lisp,
Common Lisp, SKILL++, Clojure[≈ 8 years].

For approx 8 years, I've taught Functional
Programming Applications in Scala and Clojure.

Today, I'll report some of my personal experiences
and findings.

My Teaching Philosophy

Mixed reception:

- ▶ Fundamentals are useful
- ▶ Students want to learn
- ▶ Don't waste student's time
- ▶ Think like beginner
- ▶ Make mistakes — fix mistakes
- ▶ Avoid showing only the efficient solution



We learn by doing things wrong and eventually *understanding why it's wrong*. This is challenging with a small classroom time budget.

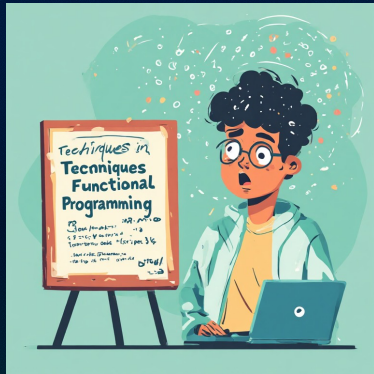
The Course



- ▶ *Techniques in Functional Programming.*
- ▶ High Level: rewrite algorithms you already know in a functional style
- ▶ 4 three-hour sessions, spring 2024, 2025, 2026.
- ▶ Mostly live coding: explain as we go, deliberately create errors live, and ask students to help figure out what went wrong.
- ▶ Grading: Quizzes and auto-graded submissions.

Why Aren't They *Looking For It*?

- ▶ Not all by choice — similar course (Scala, then Clojure) has run as both an elective and a required course.
- ▶ Electives taught in English also draw exchange students who don't speak French — regardless of interest in FP.
- ▶ The advertised title, *Techniques in Functional Programming*, never mentions Clojure or Lisp.



Syllabus

Three-hour sessions.

- ▶ **Session 1** — language overview, syntax, IDEs, recursion,
 - ▶ Simple Recursion / Tail Recursion / `loop` / `recur`
 - ▶ Using `reduce` (multiple arity)
 - ▶ Multi-threaded
- ▶ **Session 2** — higher-order functions; immutability
 - ▶ Graph Theory Applications, compute adjacency-list,
 - ▶ vectors, lists, sets, maps
 - ▶ Sequence functions: `map`, `mapcat`, `filter`, `group-by`
- ▶ **Session 3** — the `for` loop.
 - ▶ CSV parsing, plot generation (Vega plot)
- ▶ **Session 4** — concurrency.
 - ▶ Futures, atoms, examples from `@andrey.fadeev`



Assignments; Easy, Medium, Difficult, all Count the Same

- ▶ **reduce** — trivial to code. Students should think of iterative algorithms as single call to **reduce**. *Hard to understand, easy to code.*
- ▶ **factors** — recursively compute prime factors efficiently. *Easy to understand, hard to code.*
- ▶ **geodesic** — hard on *both* axes. generate a sequence of (latitude, longitude) in 2D space by divide-and-conquer; trace the shortest path on the globe between two given locations.



Assignments; Easy, Medium, Difficult, all Count the Same

- ▶ **reduce** — trivial to code. Students should think of iterative algorithms as single call to **reduce**. *Hard to understand, easy to code.*

- ▶ **factors** — recursively compute prime factors efficiently. *Easy to understand, hard to code.*

- ▶ **geodesic** — hard on *both* axes. generate a sequence of (latitude, longitude) in 2D space by divide-and-conquer; trace the shortest path on the globe between two given locations.



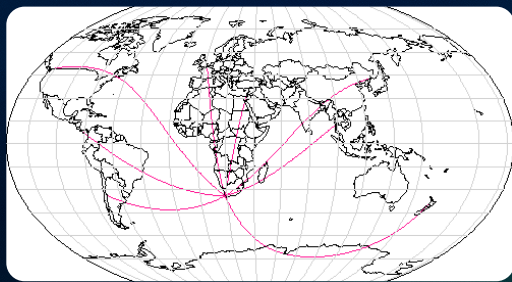
Assignments; Easy, Medium, Difficult, all Count the Same

- ▶ **reduce** — trivial to code. Students should think of iterative algorithms as single call to **reduce**. *Hard to understand, easy to code.*
- ▶ **factors** — recursively compute prime factors efficiently. *Easy to understand, hard to code.*
- ▶ **geodesic** — hard on *both* axes. generate a sequence of (latitude, longitude) in 2D space by divide-and-conquer; trace the shortest path on the globe between two given locations.



Assignments; Easy, Medium, Difficult, all Count the Same

- ▶ **reduce** — trivial to code. Students should think of iterative algorithms as single call to **reduce**. *Hard to understand, easy to code.*
- ▶ **factors** — recursively compute prime factors efficiently. *Easy to understand, hard to code.*
- ▶ **geodesic** — hard on *both* axes. generate a sequence of (latitude, longitude) in 2D space by divide-and-conquer; trace the shortest path on the globe between two given locations.



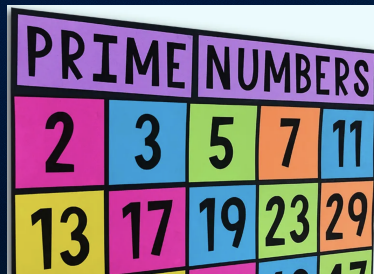
Assignments; Easy, Medium, Difficult, all Count the Same

- ▶ **reduce** — trivial to code. Students should think of iterative algorithms as single call to **reduce**. *Hard to understand, easy to code.*
- ▶ **factors** — recursively compute prime factors efficiently. *Easy to understand, hard to code.*
- ▶ **geodesic** — hard on *both* axes. generate a sequence of (latitude, longitude) in 2D space by divide-and-conquer; trace the shortest path on the globe between two given locations.



Each Assignment is a Template

```
1 (defn prime-factors [n]
2   (if (even? n)
3     (conj (prime-factors (/ n 2)) 2)
4     (loop [primes []
5           n      n
6           start  3
7           ul      (round (sqrt n))]
8       (cond (= 1 n) primes
9
10              (< ul start)
11              (throw ...)
12
13              (zero? (mod n start))
14              (recur (throw ...) ...)
15
16              :else
17              (recur (throw ...) ...))))))
```



PRIME NUMBERS				
2	3	5	7	11
13	17	19	23	29

All tests are provided.
Students **can** run them
locally.

Solution `prime-factors`

```
1  (defn prime-factors [n]
2    (if (even? n)
3      (conj (prime-factors (/ n 2)) 2)
4      (loop [primes []
5             n      n
6             start   3
7             ul       (round (sqrt n))]
8        (cond (= 1 n) primes
9
10              (< ul start) (throw ...))
11
12              ;; Therefore start <= upper-limit
13              (zero? (mod n start))
14              (recur (conj primes start)
15                     (/ n start)
16                     start
17                     (round (sqrt (/ n start))))))
18
19      :else
20      (recur (throw ...) ...))))
```

Solution `prime-factors`

```
1 (defn prime-factors [n]
2   (if (even? n)
3     (conj (prime-factors (/ n 2)) 2)
4     (loop [primes []
5            n      n
6            start  3
7            ul      (round (sqrt n))]
8       (cond (= 1 n) primes
9
10              (< ul start)
11              (throw ...)
12
13              (zero? (mod n start))
14              (recur (throw ...) ...)
15
16              ;; Therefore start is not a factor
17              :else
18              (recur primes n (+ start 2) ul))))))
```

Solution `prime-factors`

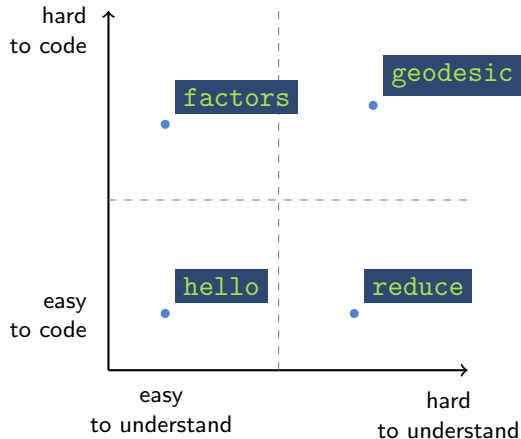
```
1  (defn prime-factors [n]
2    (if (even? n)
3      (conj (prime-factors (/ n 2)) 2)
4      (loop [primes []
5             n      n
6             start  3
7             ul      (round (sqrt n))]
8        (cond (= 1 n) primes
9
10               ;; Termination condition
11               (< ul start)
12               (conj primes (int n))
13
14               (zero? (mod n start))
15               (recur (throw ...) ...)
16
17               :else
18               (recur (throw ...) ...))))
```


Some Typical Assignments

Two axes of difficulty

- ▶ Difficulty to conceptualize
- ▶ Difficulty to implement

Every assignment counts the same.



Submit One Assignment at a Time

Recursion-2026-25-05-13-52-54
Submitted on May 25, 2026 - 13:53

Tenants > epita-ing > 2026-clojure-elective-promo-2028 > Recursion > 563bc9d6-585b-4bdf-b731-1ee2d19f1edc

100% ✓
5/5 tests passed


0ms processing time

Tests

▼ homework ✓ - 5 / 5

▼ recursion-test ✓ - 5 / 5

t-reduce ✓

t-product-doubles-loop-recur ✓

t-loop-recur ✓

t-sum-doubles-a ✓

t-product-doubles-reduce ✓

- ▶ Student submits finished exercise with `git tag ; git push`. We hereafter call this a *submission*.
- ▶ Student sees results in a web page.
- ▶ Grade is percentage of tests passed.
- ▶ HW grade is average percentage.
- ▶ **Def:** Multiple *submissions* (or tries) allowed per assignment.
- ▶ **Def:** One student's whole set of submissions for one assignment is a *student-assignment*.

Bus Factor is Unity



- ▶ Currently the support material is mainly in my head.

$$\Phi_{bus} = 1$$

- ▶ There is some documentation, but inconsistent and incomplete.
- ▶ I would like to publish the course and exercises into a reusable form.
- ▶ Possible with LLM?

See the Article

Would you like to see more example code? See the article.

The Divide



Most students don't fully "get it" by the end of the course.

A few do.

We Asked Two Questions

1. *What is the most difficult part of learning Clojure given that you are already an expert programmer in other languages?*
2. *Write a paragraph briefly comparing the Clojure language to any other programming language which you have experience with. Be free to express likes and dislikes or perceived strengths and weaknesses of the language from a beginner's point of view.*

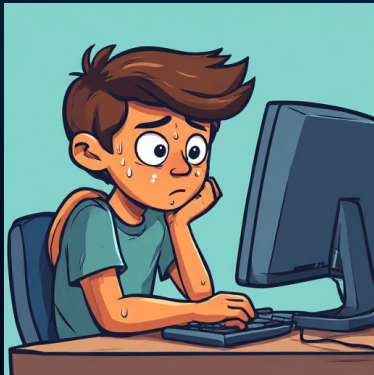


We Asked Two Questions

1. *What is the most difficult part of learning Clojure given that you are already an expert programmer in other languages?*
2. *Write a paragraph briefly comparing the Clojure language to any other programming language which you have experience with. Be free to express likes and dislikes or perceived strengths and weaknesses of the language from a beginner's point of view.*



What Fails to Land: The Data



Question: *What is the most difficult part of learning Clojure, given that you are already an expert programmer in other languages?*

Disclaimer: of 13 students, only 4 left a real answer to this question. This is not a survey with statistical power — it's a handful of real voices.

Students complain about “syntax”, what is syntax?

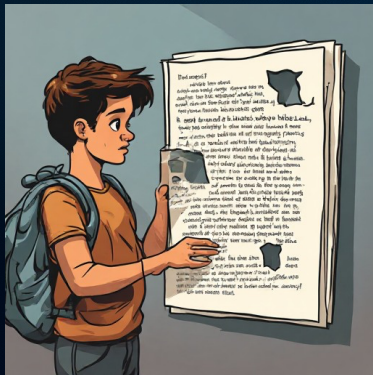
Wildly different *semantics* of “syntax”!

1. *Unfamiliar names*: “the unusual names of functions, like `conj` or `contains?` ... checking for key and not value, which makes it harder to search the docs.”
2. *The human reading experience*: “the whole syntax... makes it really hard to read and understand the code.”
3. *Literal parentheses, plus interop*: “syntax (including parenthesis)... we will have to rely on another language like Java.”



Same word. Three different obstacles.

Missing Docs, Not a Language Flaw



The “unfamiliar names” complaint came with a second half:

“... especially since the docs are incomplete for quite a few functions.”

Not everything here is about Clojure being intrinsically hard for newcomers. Some of it is ordinary documentation debt — the kind any project can chip away at, unrelated to the language itself but rather the user experience.

Rejecting the Functional Model, Not the Parentheses

The deepest instance of this complaint was about paradigm.

*“... it's a functional language,
just this makes it weird and hard.”*

Not a syntax problem. A rejection of the model itself.



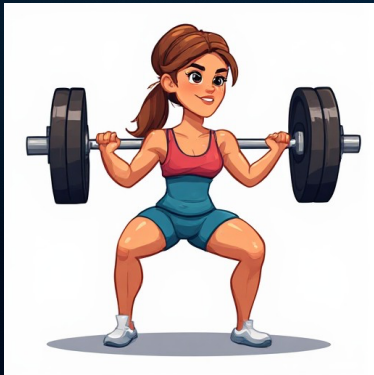
What Lands



A few students don't just tolerate the model.

They get it. Completely.

Nothing Was Difficult



Question: What is the hardest part of learning Clojure?

One student's answer was: *"Nothing."*

Question: Compare Clojure to other languages...

This student's comparison paragraph explains lazy sequence evaluation, and frames manual vs. automatic memory management as *both* a strength and a weakness — a genuinely sophisticated take.

Most Students Somewhere in Between



Understanding and struggling coexist.

- ▶ The student who found naming and docs hardest still volunteered real appreciation: the JVM ecosystem, and the `#(...)` function syntax.
- ▶ The student who found integration hardest still gave a careful, accurate four-point technical comparison to Python — parens, data-literal syntax, action-first ordering, immutability — without being asked to.

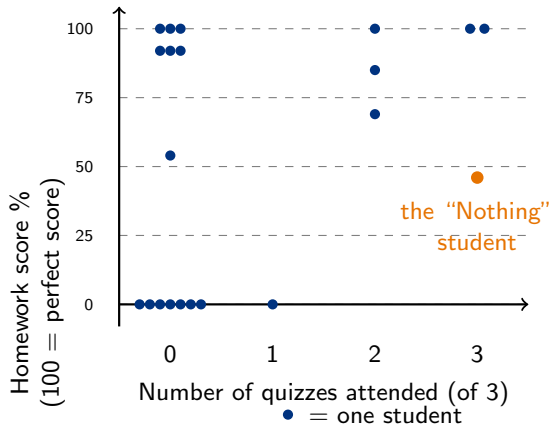
The clean cases are the exception, which is exactly why they stand out.

Same Course, Different Outcomes

Same course. Same four sessions. Same assignments.

What actually makes the difference?

Attendance Doesn't Predict the Grade (2026)



7 students never submitted a quiz — meaning, as far as attendance is measured, they never came to class.

Of those 7, 6 scored between 92% and 100% on the graded homework.

The student who said "Nothing" was the hardest part of learning Clojure? Full attendance, real understanding.

A Low Score Doesn't Mean a Student Struggled

A low homework score doesn't mean a student struggled.

You don't need 100%. Do sufficiently many problems to validate the course.

This is an elective, running concurrently with high-pressure projects. Doing every homework problem, especially the hard ones, costs time better budgeted elsewhere.

It can also mean an LLM did the work!

From the data alone, I cannot tell these apart.



Original Work or LLM?



An LLM can do all the homework. The student can avoid learning anything.

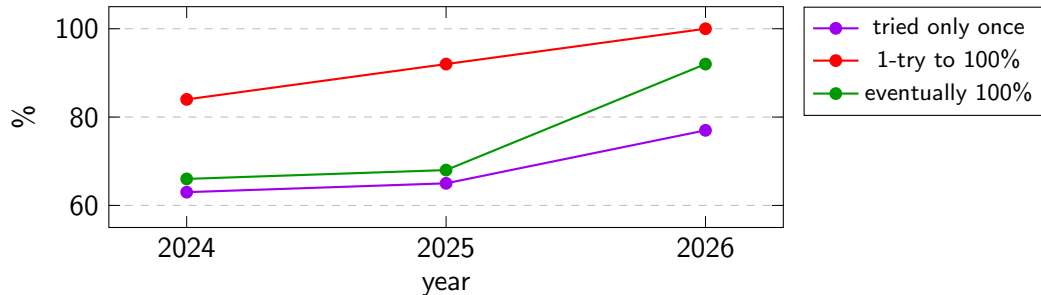
We don't know how to address this issue.

How can we know? Do we care?

What is the goal of the professor?

Every Metric, Same Direction

Three years, same auto-grading pipeline. How many attempts needed to pass?

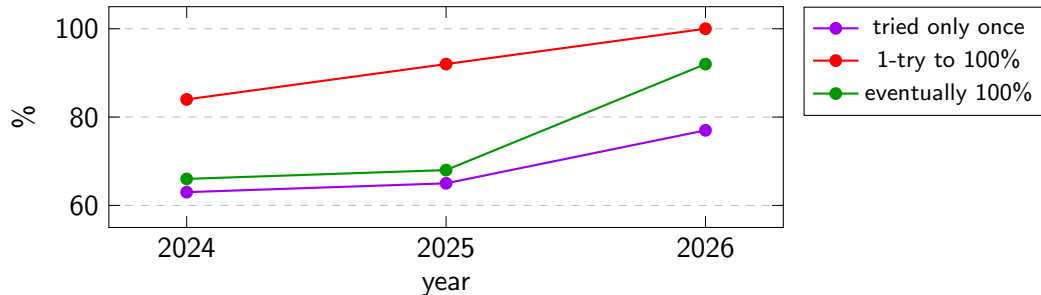


Purple curve: percentage of submissions which were tried only once.

year 2024: 214 of 339 year 2025: 182 of 279 year 2026: 121 of 158

Every Metric, Same Direction

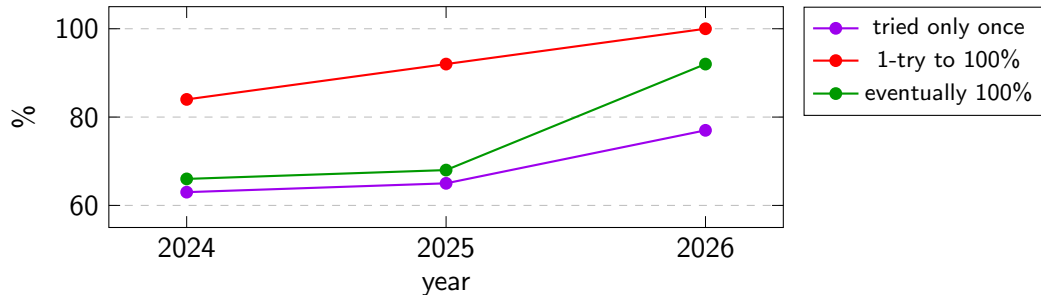
Three years, same auto-grading pipeline. How many attempts needed to pass?



Red curve: of single-try submissions, what fraction scored 100%.

Every Metric, Same Direction

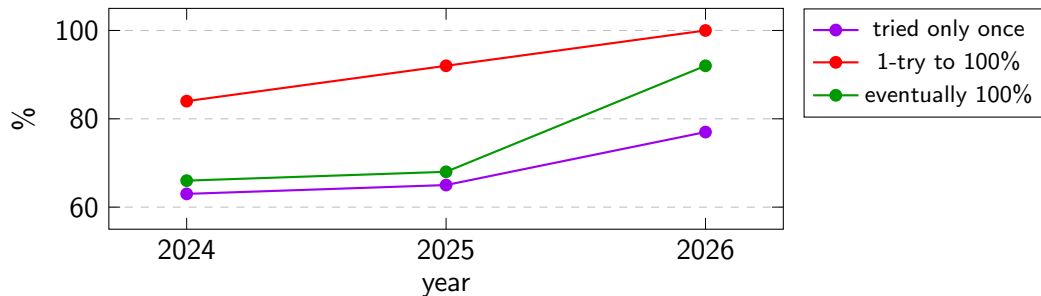
Three years, same auto-grading pipeline. How many attempts needed to pass?



Green curve: number of assignments (not students) which eventually scored 100% after possibly many attempts.

Every Metric, Same Direction

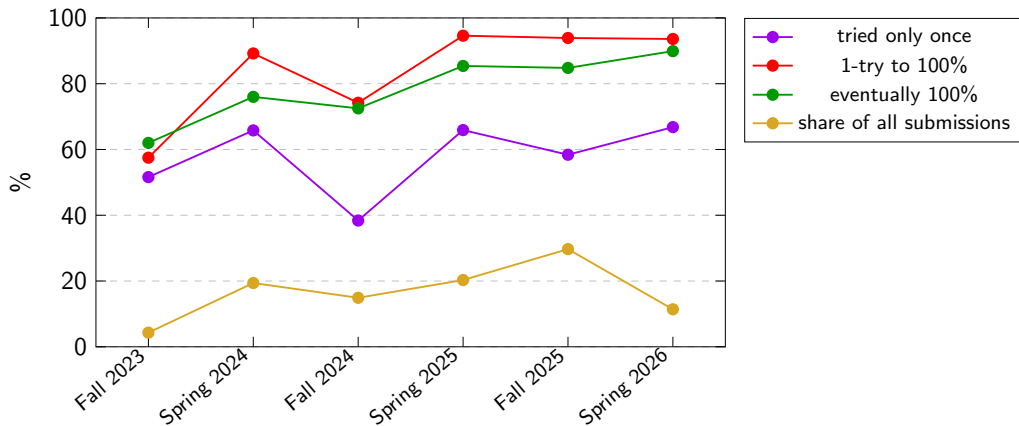
Three years, same auto-grading pipeline. How many attempts needed to pass?



This year: *every student who only tried once got it right. Zero exceptions.*

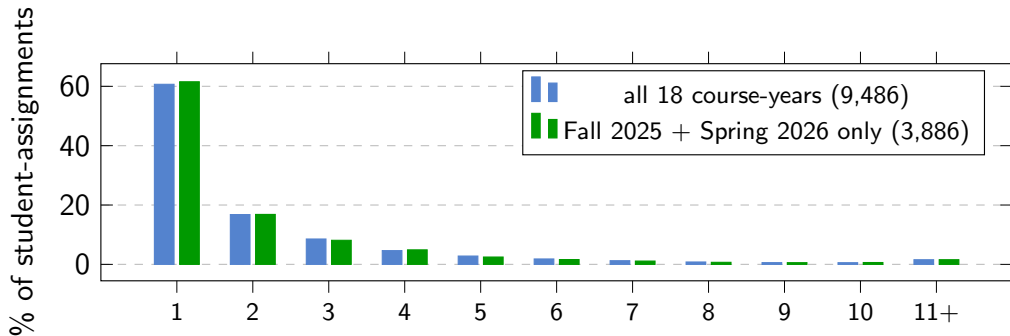
That never happened before — in 2024 and 2025, trying once and still being wrong happened regularly.

Pooled Across Seven Courses, By Calendar Time



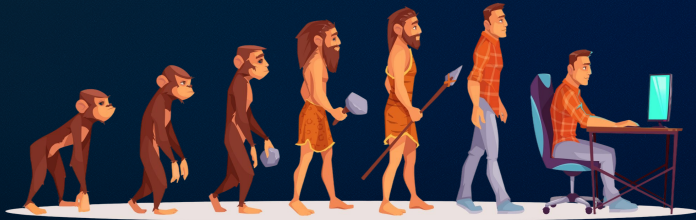
Every course's own year replaced by its actual semester; every submission weighted equally, so bigger courses count for more. 20,427 individual submissions total.

The Median Student-Assignment Needed Only One Try



A **student-assignment** = one student's attempt history on one assignment. 9,486 total, mean 2.15 tries, **median 1 try**, 60.7% solved first-try — surprising, since free resubmission should reward iterate-then-fix, not first-try success.

Correlation, Not Proof: Students Getting Smarter?



- ▶ Expected shape if smarter LLM assistance got more common, year over year.
- ▶ It is still *correlational*. Better docs, clearer assignments, a stronger cohort — none of these are ruled out.
- ▶ It cannot tell us about any one student — only about the class as a whole, over time.

We can see that SOMETHING has changed.

Perspectives

- ▶ **Partial success:** some students clearly get it – completely, and quickly. Live coding, with real errors and debugging, seems to be part of why. How to improve acceptance?
- ▶ **Real insight:** the divide between attendance and homework score is real and measurable. The three-year trend in how submissions get made is real too, and new this year specifically.
- ▶ **Increase bus-factor:** Publish the course in an electronic medium.



Open Questions



- ▶ What should we do about student use of LLMs?
- ▶ Are HW programming exercises useful, or old-fashioned?
- ▶ What does the student grade mean?
- ▶ Is learning-to-program obsolete?
- ▶ Should students study fundamentals at all?

Want to See the Course Material?



- ▶ The course repository is deliberately private, preventing students from finding solutions on GitHub.
- ▶ If you're a curious Clojurian, I can grant access on a case-by-case basis.

Get the Companion Article



This talk has a companion article —
more depth than the slides could hold.

Scan to download, or ask me for the link.

Questions?

- ▶ Email: jimka.issy@gmail.com
- ▶ Slack: Clojurians, as Jim Newton
- ▶ Discord/GitHub: [jimka2001](#)

THANK YOU, especially to
the conference organizers.

