tl;dr

`https://xpqz.github.io/learnapl/`

DYALOG 2021

x (+⌿ ×⍤¯1)⍤1 2 ⊢y

Learning APL

Search this book...

Introduction

> A language that doesn't affect the way you think about programming is not worth knowing. –*Alan Perlis*

Who is this for?

I wrote this to be the book I would have wanted to read when I started to learn APL. An introduction to APL for an experienced practitioner from a different programming language or two. We all learn in different ways, and I prefer the fundamental concepts laid bare first, and then learn by example.

I came to APL after discovering a file of solutions to the Advent of Code 2015 challenge in K, an APL derivative. That's around 100 lines of actual code, and whilst I didn't understand any of it, I kept looking at it, trying to figure out which of the 50 problems (well, 49) this was a solution to. Each of my Python solutions typically ran to 50-100 lines+ for the bulk of the problems.

Turns out it was the whole lot. That blew my mind.

What is APL?

APL is an array language, and one of the oldest programming languages still in use today, next to FORTRAN, Lisp and COBOL. APL uses its own curious-looking symbols, like `⍎⌽⍕⌈*○≡⍬`, rather than reserved words written out in English like most other languages, like C or Python. As a language, APL sits at a very high level of abstraction, making it well suited to ultra-concise formulations of algorithms.

APL is a language that time is only now beginning to catch up with. Modern processors sport dedicated vector-oriented instructions and APL presents a high degree of mechanical sympathy ideally suited to SIMD instructions and by often being completely branchless in nature. APL, and its more punk rock little sister, K, really *fly*. APL can offer unprecedented programmer efficiency, as well as all-out execution speed.

But isn't APL dead? APL is alive and well.

Contents

HELLO!

Stefan Kruger

- CS person who spent far too long at university
- Mac zealot
- Day job at planet-sized mega corp. Ken himself worked for us…
- Mostly teach customers how to stop shooting themselves in the foot
- One of those New APLers™ Morten talked about in a blog post a few months back…

APL? What's that then?

Advent of Code [About] [Events] [Shop] [Settings] [Log Out] Stefan Kruger 50*
2015 [Calendar] [AoC++] [Sponsors] [Leaderboard] [Stats]

```
25 **
24 **
23 **
22 **
21 **
20 **
19 **
18 **
17 **
16 **
15 **
14 **
13 **
12 **
11 **
10 **
 9 **
 8 **
 7 **
 6 **
 5 **
 4 **
 3 **
```

```
/ adventofcode.com  nick, ryan, peter, walter, attila, christian, ..
/ simple solutions. not necessarily shortest or fastest.

rd:{0:0N!x};tok:{" "\:'rd x};lst:{{.*|x}'tok x};col:{(tok y)[;x]}         / read tokens lastvals columns
vs:{(1_r)-x*-1_r:|(_%)\(,y),|x};odo:{vs[x;!*/x]};sets:{odo x#2}          / decode odometer powerset
prm:{x{,/(>:'t=/:t:*x)@\:x:0,'1+x}/,!0}                                  / permutations
cmb:{$[0>@y;,/(+,+,!y-:x){(x+z){,/x,''y}'x#\:y}[1+!y]/!x-:1;y cmb[x]@#y]} / combinations

x:1 -1"()"?*rd`p1
+/x
1+(+\x)?-1

x:.:''"x"\:'rd`p2
+/{+/7#x*1_x,*x}'x:{x@<x}'x
+/{(*/x)+2*+/2#x}'x

x:"^>v<"?*rd`p3
#?0,+\a:(x,-x:1,#x)x
#?0,/++\0N 2#a

x:*rd`p4
(15<16/:3#-15!x,$:)(1+)/1
(16/:3#-15!x,$:)(1+)/1

x:rd`p5
+/{(2<+/x in"aeiou")&(|/=':x)[illegible]/(p[illegible]in$[illegible]`dc`qp[illegible]x}[illegible]
+/{(|/(-2_p)in'(!#r)[illegible]\:r[illegible]:,[illegible])&[illegible](-[illegible]x)=2[illegible]}'[illegible]

x:{.:'(|x)2 0}[illegible]k`p[illegible]
f:"fn "?(0:`p6)[illegible]6;x[illegible]{x+[illegible]:[illegible]-x}/'x
+//./[1000 1000#[illegible]x;([illegible];1|;1-)f]
+//./[1000 1000#[illegible];x;(0|-1+;1+;2+)f]

x:((*|:)'x)!x:tok`p7
A:&;O:|;L:{y_x,0>!y:0b/:y};R:{|L[|x;y]};e:{$["a">*x;0b\:. x;@a:d x;a;d[x]:$[3=#a;e@*a;4=#a;~e a 1;(.*a 1)/e'a 0 2]]}
d:x;0b/:r:e@,"a"
d:x;d[,"b"]:r;0b/:e@,"a"

x:rd`p8
+/{2+/(1+2*"x"=1_x)*<\"\\"=-1_x}'x
+/{2+/x in"\"\\"}'x

m|:+m:@[;!8]'(0,+\!7)_|lst`p9
&/x:(+/m':)'prm 8
|/x

x:*rd`p10
#40{,/{($#x),*x}'(&~=':x)_x}/x
#50{,/{($#x),*x}'(&~=':x)_x}/x

x:-97+"j"$*rd`p11 / (run of three;~any"oil"in;2 doubles)
f:&/(|/&':1=-'::~|/8 11 14 in:1<+/<\=':)@\::n:vs[8#26]1+26/:
```

IMPOSSIBLE!

k?
~100 loc

ALL OF THEM?

K is sorta APL+Lisp...?

APL: the...

of array langs

DYALOG
2021

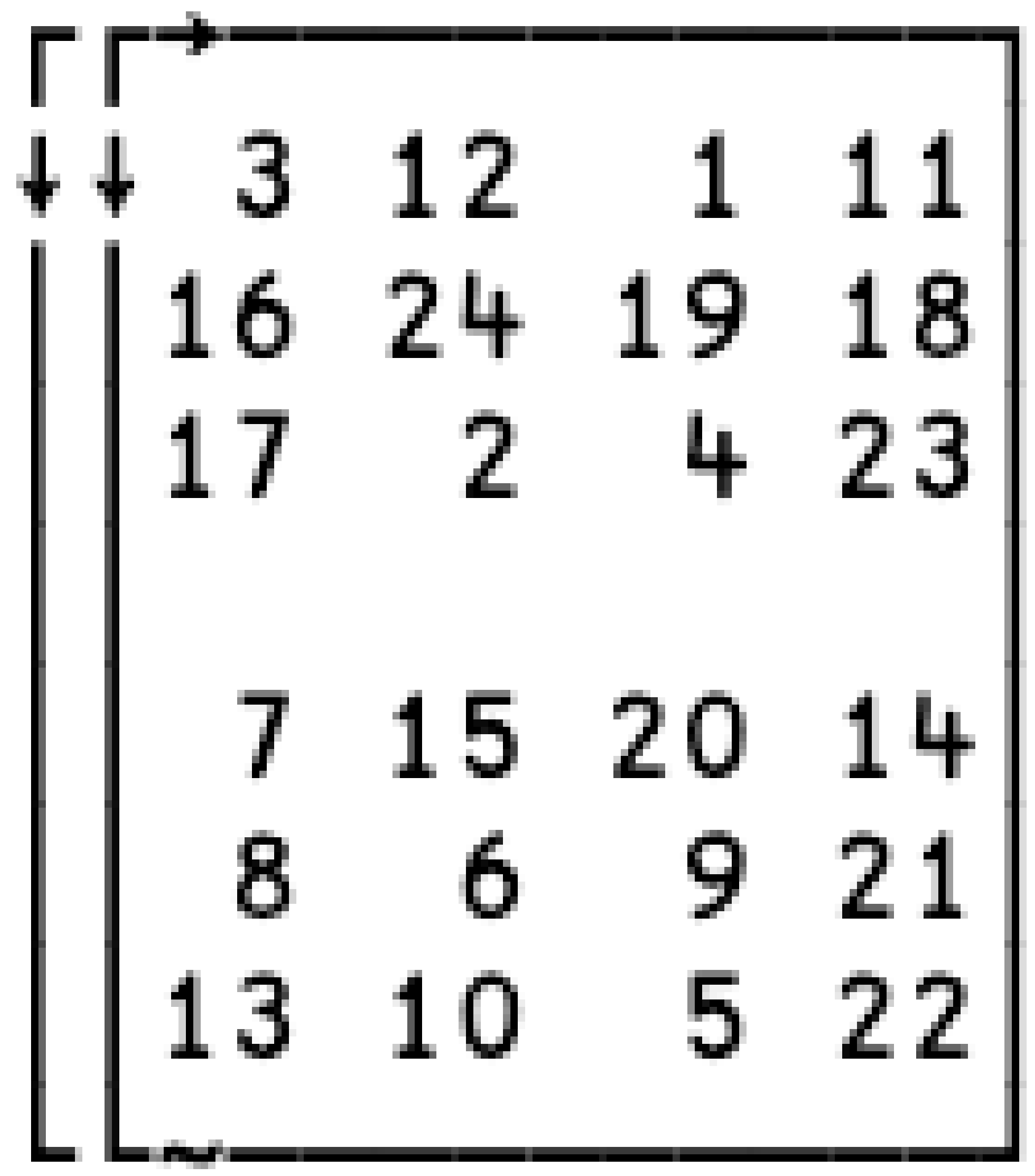
⍨ ⍤ ⍥ ⊢ ⍉ ⍣ ∇ ← ⊖

⍨

APL

Ingimundr ok Þialfi þæiʀ ræistu stæin þenna at Þōrkætil, faður sinn.

Ingimundr ok Þialfi þeiʀ ræistu stæin þenna at Þōrkætil, faður sinn.



3	12	1	11
16	24	19	18
17	2	4	23
7	15	20	14
8	6	9	21
13	10	5	22

APL has a reputation:

It's for mathematicians

Unreadable

Write-only

Impossible to learn

Community is a Council of Wizards

One of these is true

The Council of Wizards

The APL Orchard

Isn't it dead?

...or at best a living fossil?

“Old technologies that have stuck around are *sharks*, not dinosaurs. They solve problems so well that they have survived the rapid changes that occur constantly in the technology world. Don't bet against them.

–*Justin Etheredge*

DYALOG 2021

“Old technologies that have stuck around are *sharks*, not dinosaurs. They solve problems so well that they have survived the rapid changes that occur constantly in the technology world. Don't bet against them.

–Justin Etheredge

DYALOG 2021

“ Good APL follows a set of best practices that directly contradict and conflict with traditional programming wisdom. Indeed, APL design patterns appear as Anti-patterns in most other programming languages.

–Aaron Hsu

DYALOG 2021

This cognitive dissonance is one reason why some “computer people” *hate* APL.

Edsger Dijkstra

For me, it's the main draw

Learning APL ...is an act of

REBELLION

Learning APL

Not hard:

Learning to type

What glyphs mean

Coding right to left

Hard:

Data-parallel problem solving; thinking in arrays

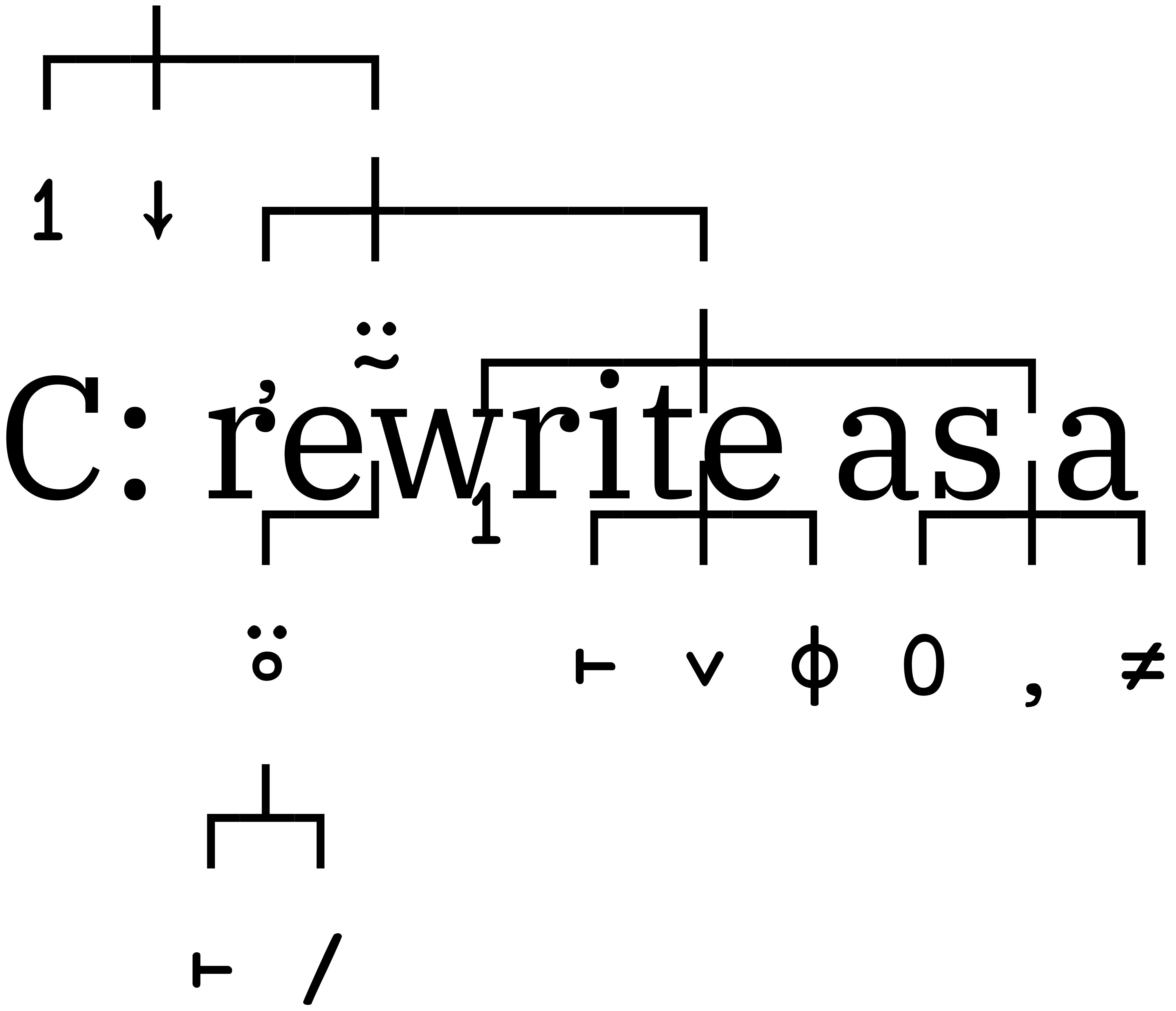
Performance intuition

Reading tacit code

```
1↓,⊢⍤/⍨1(⊢∨⌽)0,≠
```

From APLCart

```
1↓,⊢⍤/⍨1(⊢∨⌽)0,≠
```



CMC: rewrite as a dfn

```
1↓,⊢⍤/⍨1(⊢∨⌽)0,≠
```

APL is a beautiful thing!

```
1↓,⊢⍤/⍨1(⊢∨⌽)0,≠
```

Why are there so few books?

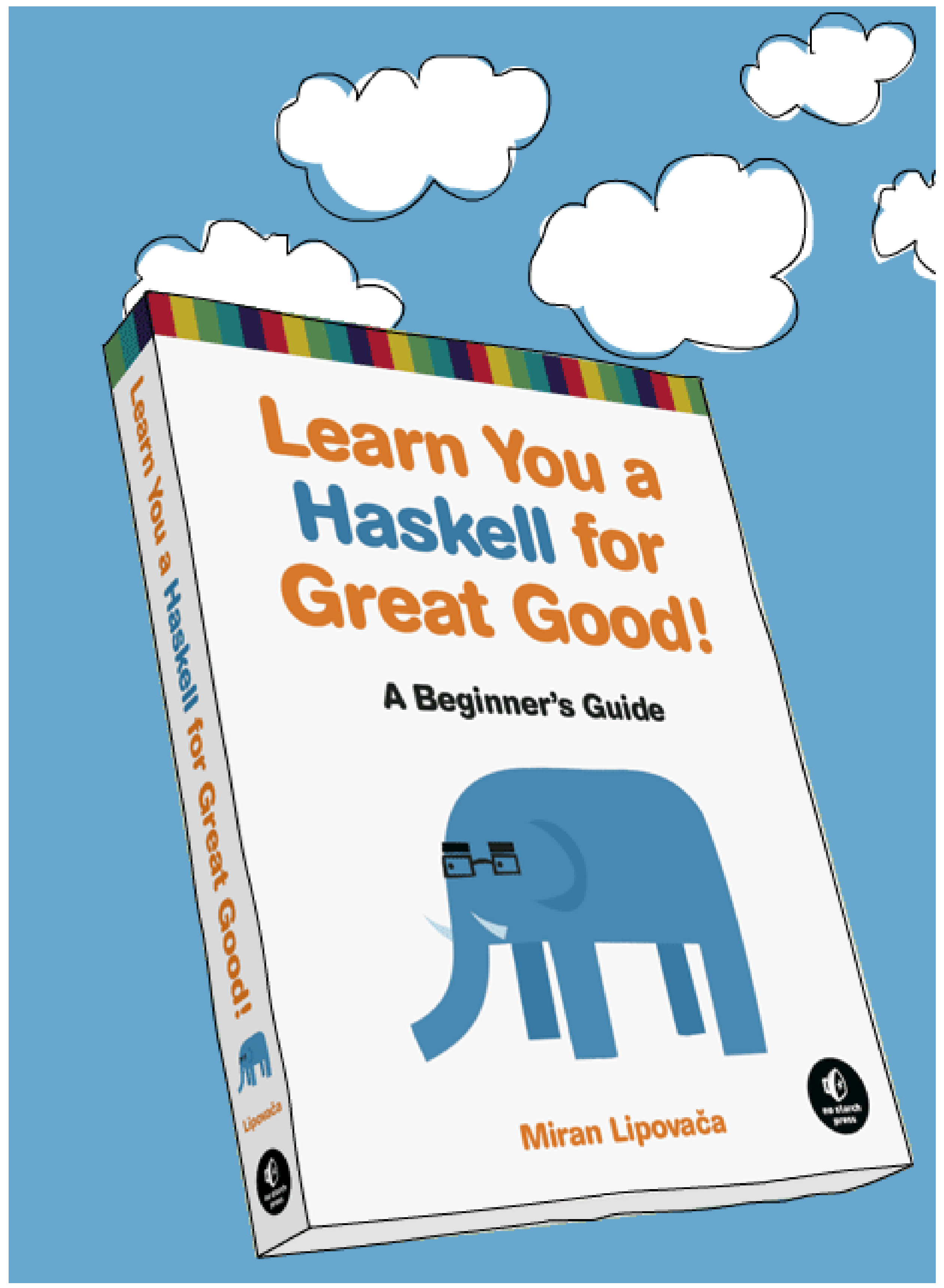
Why are there so few books?

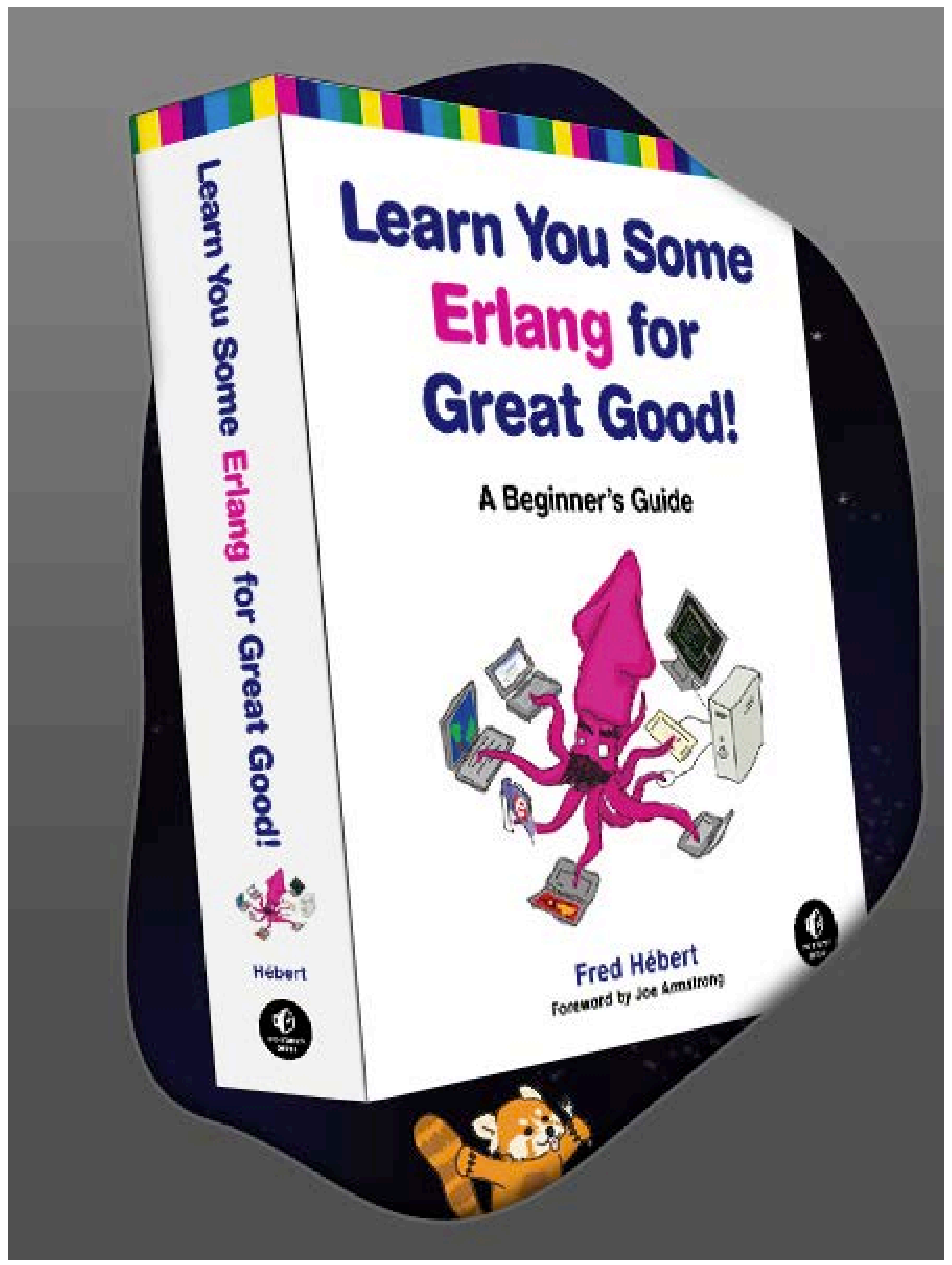
- I know how to program
- I am impatient
- Show me examples! I'll figure out how they work
- MDAPL: at the time outdated, too long, too "trad" (IMHO!)

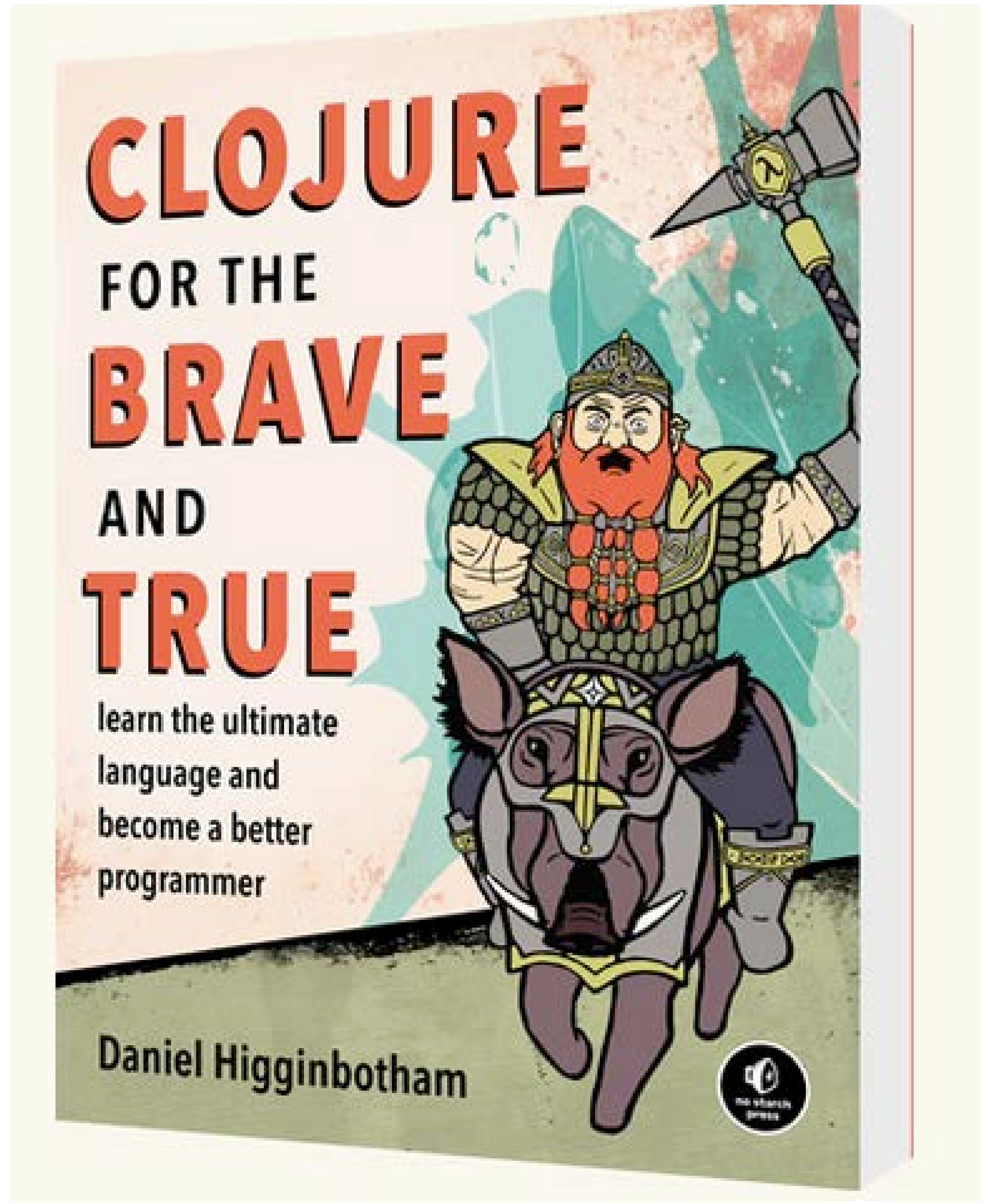
➡ Kudos to Rodrigo for bringing MDAPL into the modern era!

Something like these

APL




Learn You Some Erlang for Great Good!
A Beginner's Guide
Fred Hébert
Foreword by Joe Armstrong
Hébert


CLOJURE FOR THE BRAVE AND TRUE
learn the ultimate language and become a better programmer
Daniel Higginbotham

Can I do something about this?

(Disregarding the fact I barely know the language)

"Ask not what APL can do for you...

...ask what you can do for APL"

The Plan

```
⎕IO←0
```

- Task 0: Plow through the back catalog of APL Orchard "Cultivations"
- Task 1: read every APL-related paper I can find by Roger Hui
- Tasks 2015 - 2020: do all previous *Advent of Code* problem sets in APL -- this took me best part of a year... I do have a day job!
- Task ??: start writing, learn on the job, hope for the best

What did I learn?

Well, I picked up APL

I've become a better programmer.

APL fits my mindset

but then again, my other hobby is regular expressions

(dramatic pause)

What could be better?

Biased observations from a "New APLer" perspective

…on a Mac

It's awkward to fit Dyalog into a modern* workflow

*emacs, 1978; vim, 1991; git, 2005; VS Code, 2015

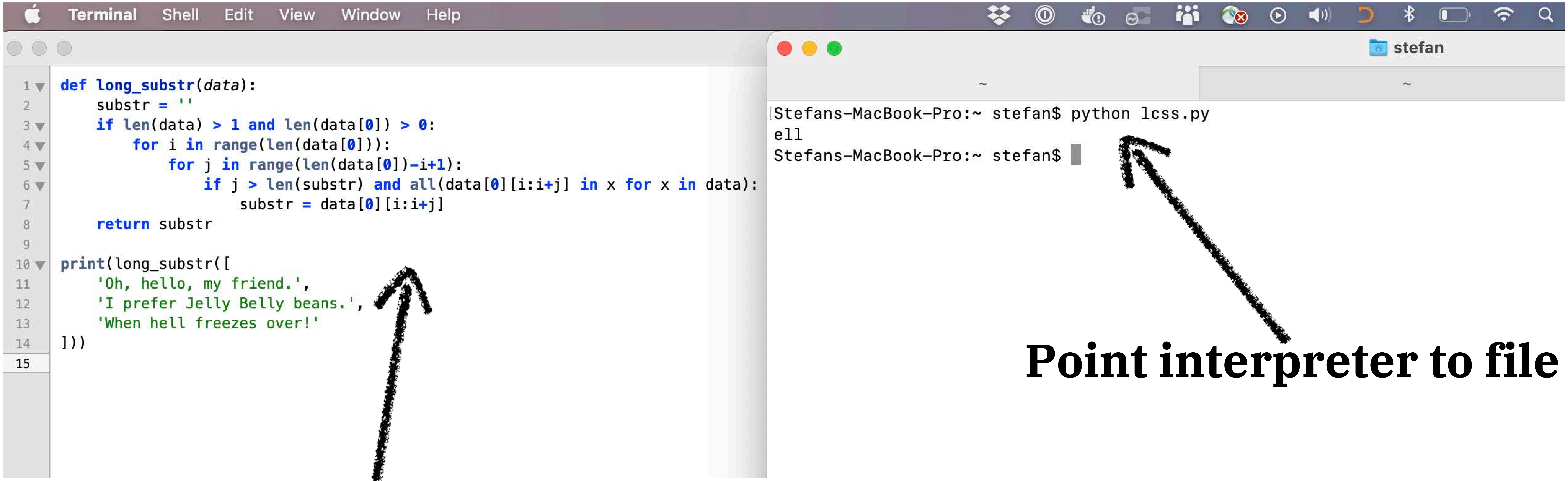
WHAT IF I TOLD YOU
CODE CAN BE STORED IN TEXT FILES

Don't get me wrong: it can be done

To work with code as text

- Install a completely separate application stack, Microsoft's .NET Core
- Install latest version of the LINK library (tricky on a Mac)
- Use LINK to map a directory on the disk containing my code to a namespace
- To execute a text file from the terminal command line…. (pre-18.1/2). On a Mac… good luck
- To be fair, once set up, it works

Other languages:



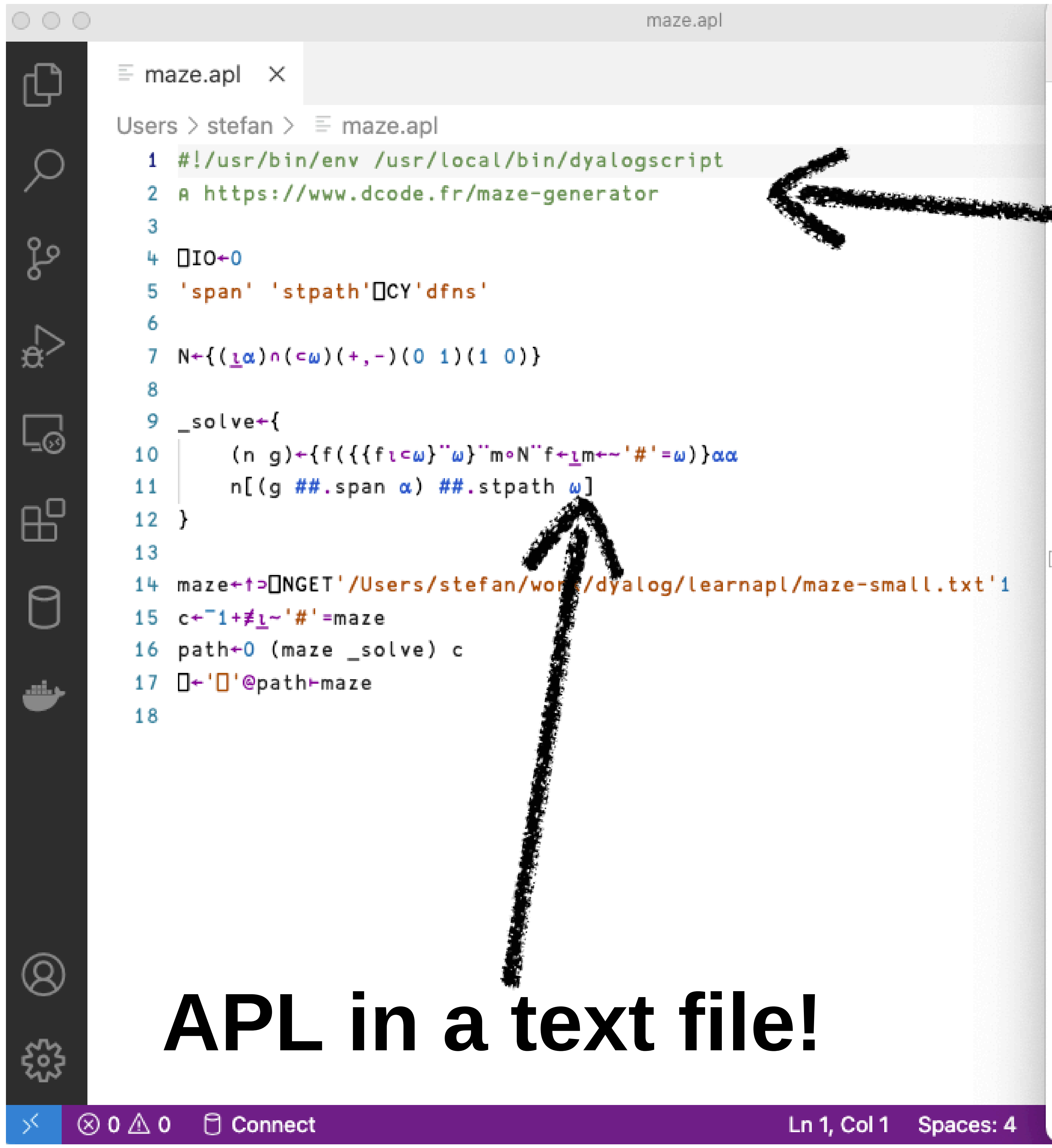
```python
def long_substr(data):
    substr = ''
    if len(data) > 1 and len(data[0]) > 0:
        for i in range(len(data[0])):
            for j in range(len(data[0])-i+1):
                if j > len(substr) and all(data[0][i:i+j] in x for x in data):
                    substr = data[0][i:i+j]
    return substr

print(long_substr([
    'Oh, hello, my friend.',
    'I prefer Jelly Belly beans.',
    'When hell freezes over!'
]))
```

```
Stefans-MacBook-Pro:~ stefan$ python lcss.py
ell
Stefans-MacBook-Pro:~ stefan$
```

Point interpreter to file

Code in a text file

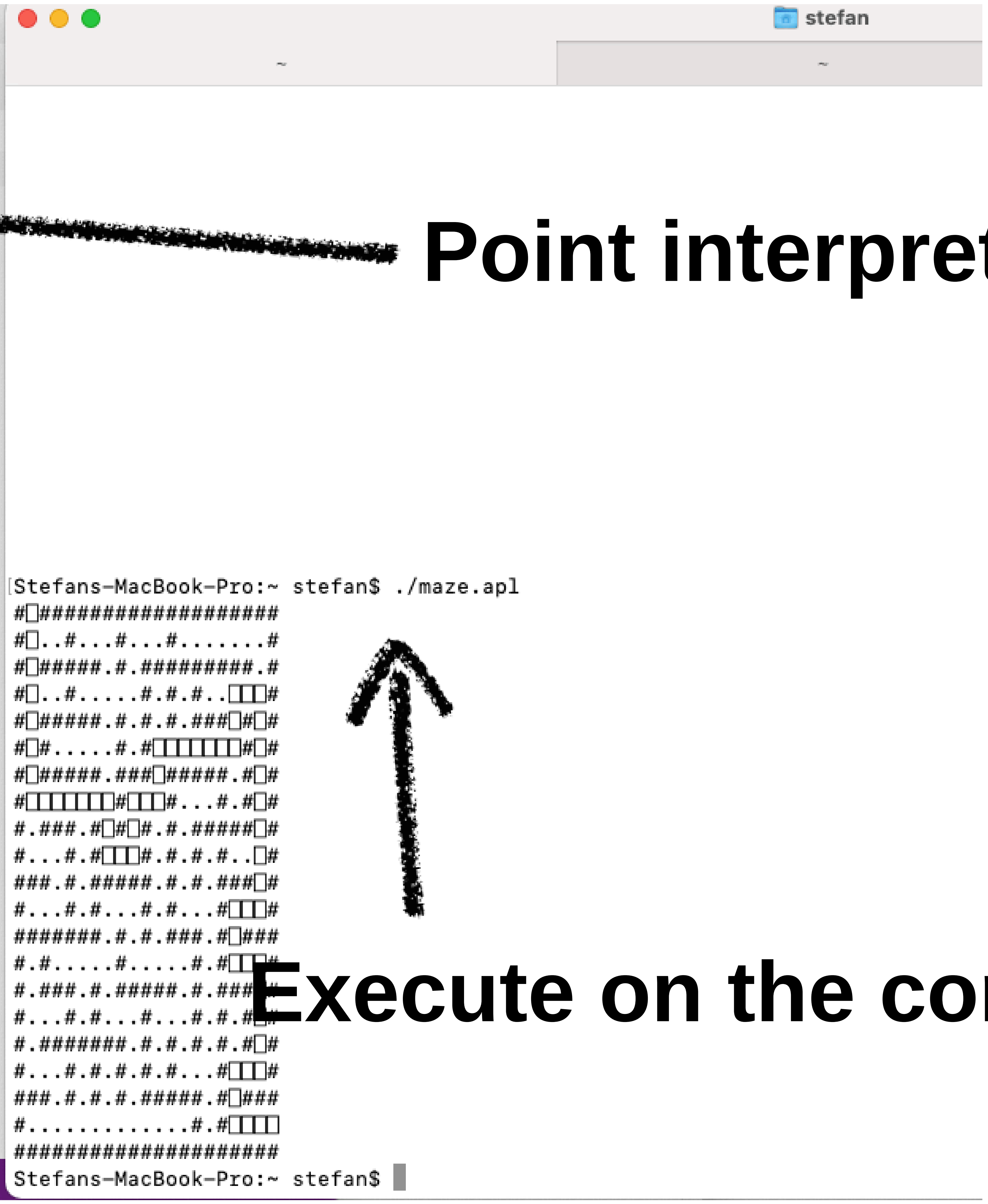
Great things are coming!



```
#!/usr/bin/env /usr/local/bin/dyalogscript
⍝ https://www.dcode.fr/maze-generator

⎕IO←0
'span' 'stpath'⎕CY'dfns'

N←{(⍸⍺)∩(⊂⍵)(+,-)(0 1)(1 0)}

_solve←{
    (n g)←{f({{f⍳⊂⍵}¨⍵}¨m∘N¨f←⍸m←~'#'=⍵)}⍺⍺
    n[(g ##.span ⍺) ##.stpath ⍵]
}

maze←↑⊃⎕NGET'/Users/stefan/work/dyalog/learnapl/maze-small.txt'1
c←¯1+⍸⍸~'#'=maze
path←0 (maze _solve) c
⎕←'⎕'@path⊢maze
```

Point interpreter to file

APL in a text file!



```
Stefans-MacBook-Pro:~ stefan$ ./maze.apl
#⎕###################
#⎕..#...#...#.......#
#⎕#####.#.#########.#
#⎕..#.....#.#.#..⎕⎕⎕#
#⎕#####.#.#.#.###⎕#⎕#
#⎕#.....#.#⎕⎕⎕⎕⎕⎕⎕#⎕#
#⎕#####.###⎕#####.#⎕#
#⎕⎕⎕⎕⎕⎕⎕#⎕⎕⎕#...#.#⎕#
#.###.#⎕#⎕#.#.#####⎕#
#...#.#⎕⎕⎕#.#.#.#..⎕#
###.#.#####.#.#.###⎕#
#...#.#...#.#...#⎕⎕⎕#
#######.#.#.###.#⎕###
#.#.....#.....#.#⎕⎕⎕#
#.###.#.#####.#.###⎕#
#...#.#...#...#.#.#⎕#
#.#######.#.#.#.#.#⎕#
#...#.#.#.#.#...#⎕⎕⎕#
###.#.#.#.#####.#⎕###
#.............#.#⎕⎕⎕⎕
#####################
Stefans-MacBook-Pro:~ stefan$
```

Execute on the command line

MIND BLOWN

Make it easier for me to use tools I already use

Be better at Mac

It's a *nix workstation

Yes, *really*.

Give me a tarball, not an "app"

Treat the Mac exactly like Linux!

Final thoughts

apl

Writing a book is an excellent way of *learning* APL

DYALOG 2021

If a single person finds it useful, it was worth it

I'm not a traditional Dyalog user

I don't build big GUI-driven applications in OO-APL

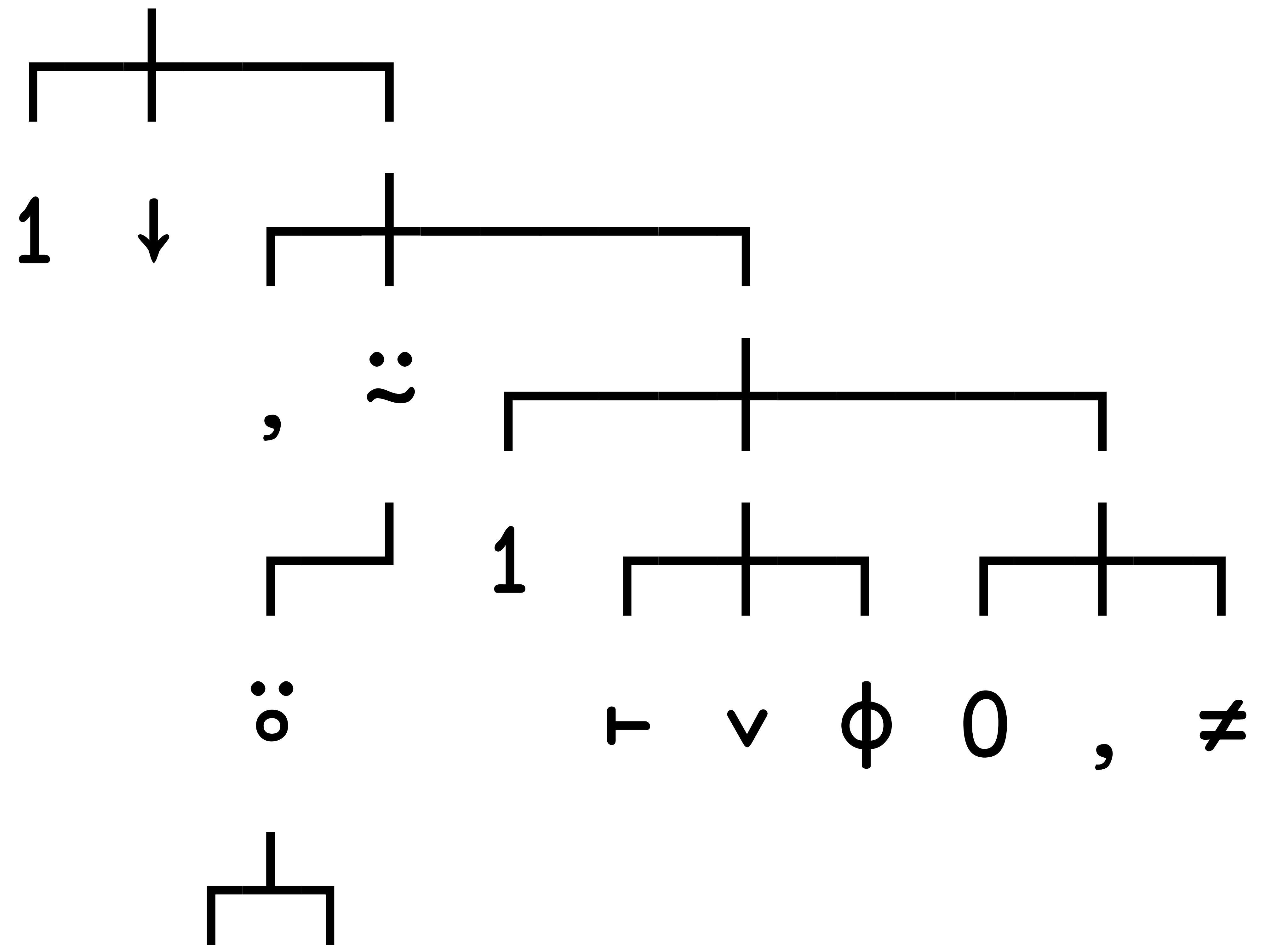
Not a "domain expert"

I hope that Dyalog sees a future for APL as a tool also for the rest of us:

...coders, toolsmiths, data analysts, ad-hoc scripters in polyglot environments who see APL as a refuge from the ills of OO

An elegant weapon for a more civilized age

```
1↓,⊢⍤/⍨1(⊢∨⌽)0,≠
```



```
{1↓(x∨1⌽x←0,⍺≠⍵)/⍺,⍵}
```

```
1↓,⊢⍤/⍨1(⊢∨⌽)0,≠
{1↓(x∨1⌽x←0,⍺≠⍵)/⍺,⍵}
```

My thanks to Rodrigo and Rory who kindly helped with proofing

DYALOG 2021

*any remaining errors or lies are mine alone and the whole lot at the APL Orchard, without whom, etc `https://xpqz.github.io/learnapl/`

Thank you

DYALOG 2021