DYALOG

Dyalog APL: What It Is and How It Works?

Aarush Bhat

Dyalog Ltd.

Coming up next...

APL

A Programming Language

What is APL?

Symbolic

Array-oriented Programming Language

for Communicating Algorithms

to Computers

and Humans

What is APL?

Symbolic

What is APL?

Symbolic

Array-oriented Programming Language

```
(5,6,7,8)
```

What is APL?

Symbolic

Array-oriented Programming Language

```
      2+(5,6,7,8)
7 8 9 10
```

What is APL?

Symbolic

Array-oriented Programming Language

```
2↓(5,6,7,8)
```

```
7 8
```

What is APL?

Symbolic

Array-oriented Programming Language

```
2↑(5,6,7,8)
```

```
5 6
```

What is APL?

Symbolic

Array-oriented Programming Language

```
      ⌽(5,6,7,8)
8 7 6 5
```

What is APL?

Symbolic

Array-oriented Programming Language

```
      table
┌─┬─┬─┐
│T│r│y│
├─┼─┼─┤
│A│P│L│
├─┼─┼─┤
│n│o│w│
└─┴─┴─┘
```

```
      ⌽table
┌─┬─┬─┐
│y│r│T│
├─┼─┼─┤
│L│P│A│
├─┼─┼─┤
│w│o│n│
└─┴─┴─┘
```

What is APL?

Symbolic

Array-oriented Programming Language

```
table
```

T	r	y
A	P	L
n	o	w

```
2↓⌽table
```

w	o	n

What is APL?

Symbolic

Array-oriented Programming Language

```
table
```

T	r	y
A	P	L
n	o	w

```
2 1↓⌽table
```

o	n

What is APL?

Symbolic

Array-oriented Programming Language

```
table
```

T	r	y
A	P	L
n	o	w

```
⍉2 1↓⌽table
```

o
n

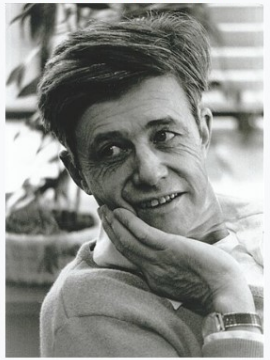
Kenneth E. Iverson

Article Talk

From Wikipedia, the free encyclopedia

Kenneth Eugene Iverson (17 December 1920 – 19 October 2004) was a Canadian computer scientist noted for the development of the programming language APL. He was honored with the Turing Award in 1979 "for his pioneering effort in programming languages and mathematical notation resulting in what the computing field now knows as APL; for his contributions to the implementation of interactive systems, to educational uses of APL, and to programming language theory and practice".[1]

Kenneth E. Iverson



Born	17 December 1920 Camrose, Alberta, Canada
Died	19 October 2004 (aged 83) Toronto, Canada
Alma mater	Queen's University Harvard University
Known for	Programming languages: APL, J
Awards	IBM Fellow Harry H. Goode Memorial Award Turing Award Computer Pioneer Award

Symbols, symbols and symbols

Myth: “APL is Unreadable”

$$(\times/!\,x-1)\div!\,(+/x)-1$$

$$\frac{\prod_{i=1}^{n}(x_i-1)!}{\left(\left(\sum_{i=1}^{n}x_i\right)-1\right)!}$$

数学函数

Математичка функција

دالة رياضية

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)

Sum=x=>x.reduce((a,b)=>a+b,0)
Prd=x=>x.reduce((a,b)=>a*b,1)
Rng=x=>[...Array(x).keys()]
Fac=x=>Prd(Rng(x+1).slice(1))
Prd(x.map(e=>Fac(e-1)))/Fac(Sum(x)-1)
```

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)
```

```
Sum = x => x.reduce((a, b) => a + b, 0)          Sum ← +/
Prd = x => x.reduce((a, b) => a * b, 1)          Prd ← ×/
Rng = x => [... Array(x).keys()]                 Rng ← ⍳
Fac = x => Prd(Rng(x + 1).slice(1))              Fac ← Prd 1 ↓ (Rng +∘1)
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)    (Prd Fac¨x - 1) ÷ Fac(Sum x) - 1
```

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)
```

```
Sum = x => x.reduce((a, b) => a + b, 0)
Prd = x => x.reduce((a, b) => a * b, 1)
Rng = x => [... Array(x).keys()]
Fac = x => Prd(Rng(x + 1).slice(1))
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)
```

```
Sum ← +/
Prd ← ×/
Rng ← ⍳
Fac ← Prd 1 ↓ (Rng ∘.+⍨1)
(Prd Fac¨x - 1) ÷ Fac(Sum x) - 1
```

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)
```

```
Sum = x => x.reduce((a, b) => a + b, 0)
Prd = x => x.reduce((a, b) => a * b, 1)
Rng = x => [... Array(x).keys()]
Fac = x => Prd(Rng(x + 1).slice(1))
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)
```

```
Sum ← +/
Prd ← ×/
Rng ← ⍳
Fac ← Prd 1 ↓ (Rng +∘1)
(Prd Fac¨x - 1) ÷ Fac(Sum x) - 1
```

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)
```

```
Sum = x => x.reduce((a, b) => a + b, 0)
Prd = x => x.reduce((a, b) => a * b, 1)
Rng = x => [... Array(x).keys()]
Fac = x => Prd(Rng(x + 1).slice(1))
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)
```

```
Sum ← +/
Prd ← ×/
Rng ← ⍳
Fac ← Prd 1 ↓ (Rng +∘1)
(Prd Fac¨x - 1) ÷ Fac(Sum x) - 1
```

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)
```

```
Sum = x => x.reduce((a, b) => a + b, 0)
Prd = x => x.reduce((a, b) => a * b, 1)
Rng = x => [... Array(x).keys()]
Fac = x => Prd(Rng(x + 1).slice(1)
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)
```

```
Sum ← +/
Prd ← ×/
Rng ← ⍳
Fac ← Prd 1 ↓ (Rng ∘.+⍨1)
(Prd Fac¨x - 1) ÷ Fac(Sum x) - 1
```

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)
```

```
Sum = x => x.reduce((a, b) => a + b, 0)
Prd = x => x.reduce((a, b) => a * b, 1)
Rng = x => [... Array(x).keys()]
Fac = x => Prd(Rng(x + 1).slice(1))
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)
```

```
Sum ← +/
Prd ← ×/
Rng ← ⍳
Fac ← Prd 1 ↓ (Rng ∘.+⍨1)
(Prd Fac¨x - 1) ÷ Fac(Sum x) - 1
```

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)
```

```
Sum = x => x.reduce((a, b) => a + b, 0)
Prd = x => x.reduce((a, b) => a * b, 1)
Rng = x => [... Array(x).keys()]
Fac = x => Prd(Rng(x + 1).slice(1))
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)
```

```
Sum ← +/
Prd ← ×/
Rng ← ⍳
Fac ← Prd 1 ↓ (Rng ∘.+1)
(Prd Fac x - 1) ÷ Fac(Sum x) - 1
```

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)
```

```
Sum = x => x.reduce((a, b) => a + b, 0)          Sum ← +/
Prd = x => x.reduce((a, b) => a * b, 1)          Prd ← ×/
Rng = x => [... Array(x).keys()]
Fac = x => Prd(Rng(x + 1).slice(1))              Fac ← !
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)    (Prd Fac x - 1) ÷ Fac(Sum x) - 1
```

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)
```

```
Sum = x => x.reduce((a, b) => a + b, 0)
Prd = x => x.reduce((a, b) => a * b, 1)
Rng = x => [... Array(x).keys()]
Fac = x => Prd(Rng(x + 1).slice(1))
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)
```

```
Sum ← +/
Prd ← ×/

Fac ← !
(Prd Fac x - 1) ÷ Fac(Sum x) - 1
```

Myth: “APL is Unreadable”

```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)

Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)

Sum = x => x.reduce((a, b) => a + b, 0)
Prd = x => x.reduce((a, b) => a * b, 1)
Rng = x => [... Array(x).keys()]
Fac = x => Prd(Rng(x + 1).slice(1))
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)  ( ×/  !  x - 1) ÷  ! ( +/ x) - 1
```

Myth: “APL is Unreadable”

$$(\times/!x-1)\div!(+/x)-1$$

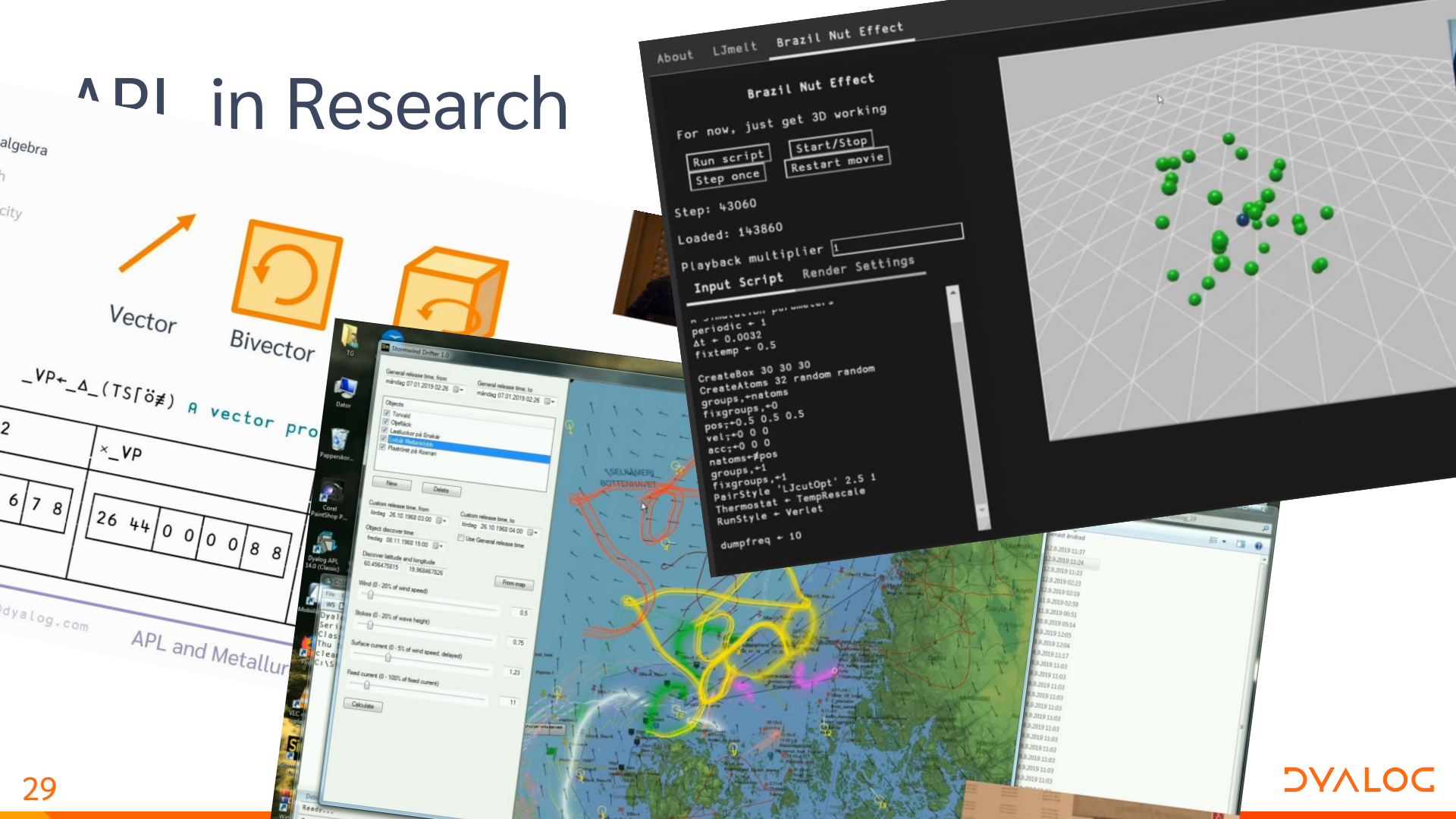
```
x.map(e=>[...Array(e).keys()].slice(1).reduce((a,b)=>a*b,1)).reduce((a,b)=>a*b)
  /[...Array(x.reduce((a,b)=>a+b)).keys()].slice(1).reduce((a,b)=>a*b,1)
```

```
Fac=x=>[...Array(x+1).keys()].slice(1).reduce((a,b)=>a*b,1)
x.map(e=>Fac(e-1)).reduce((a,b)=>a*b)/Fac(x.reduce((a,b)=>a+b)-1)
```

```
Sum = x => x.reduce((a, b) => a + b, 0)
Prd = x => x.reduce((a, b) => a * b, 1)
Rng = x => [... Array(x).keys()]
Fac = x => Prd(Rng(x + 1).slice(1))
Prd(x.map(e => Fac(e - 1))) / Fac(Sum(x) - 1)  (×/!x-1)÷!(+/x)-1
```

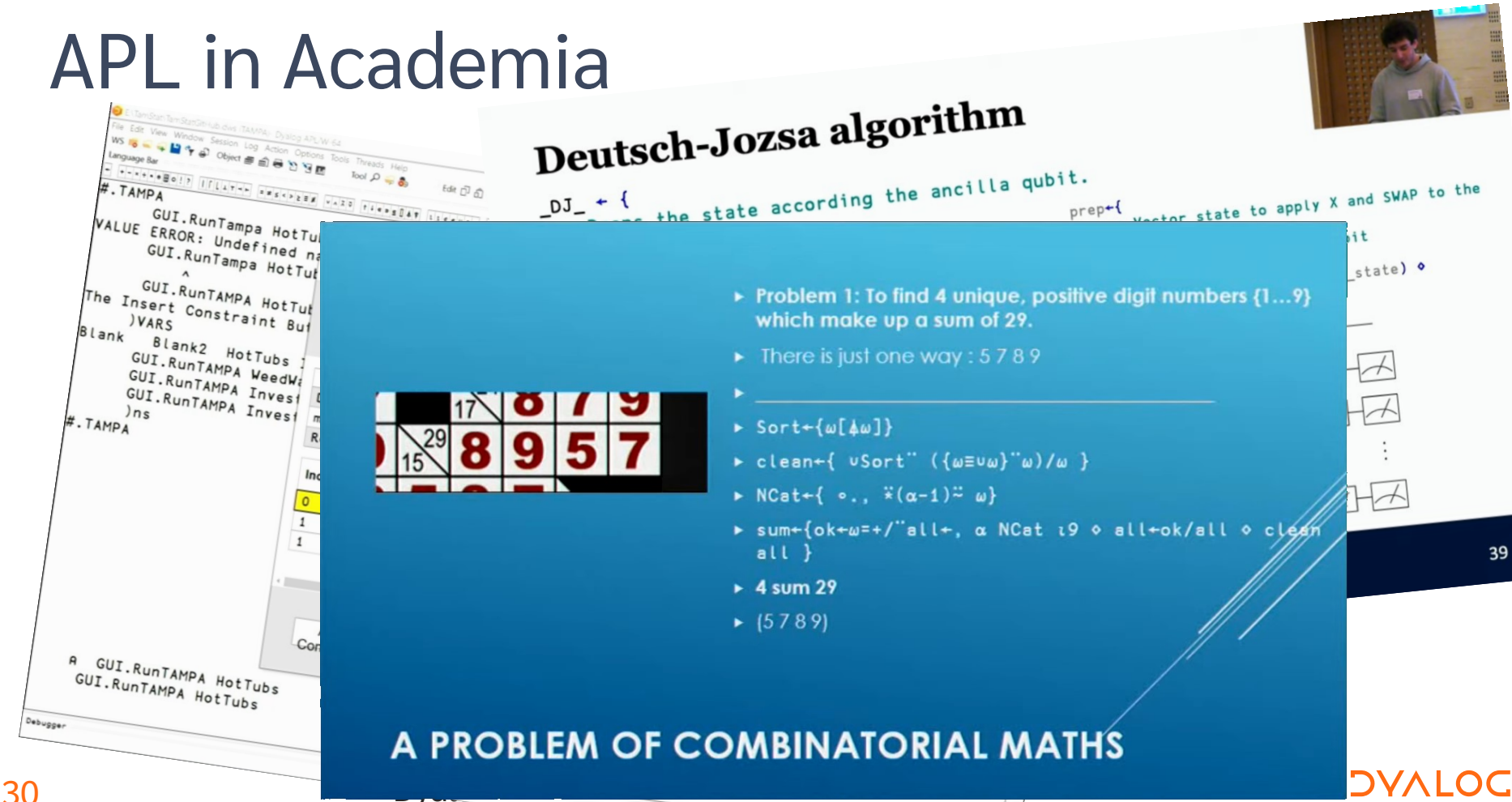
But, where is this used?

APL in Research

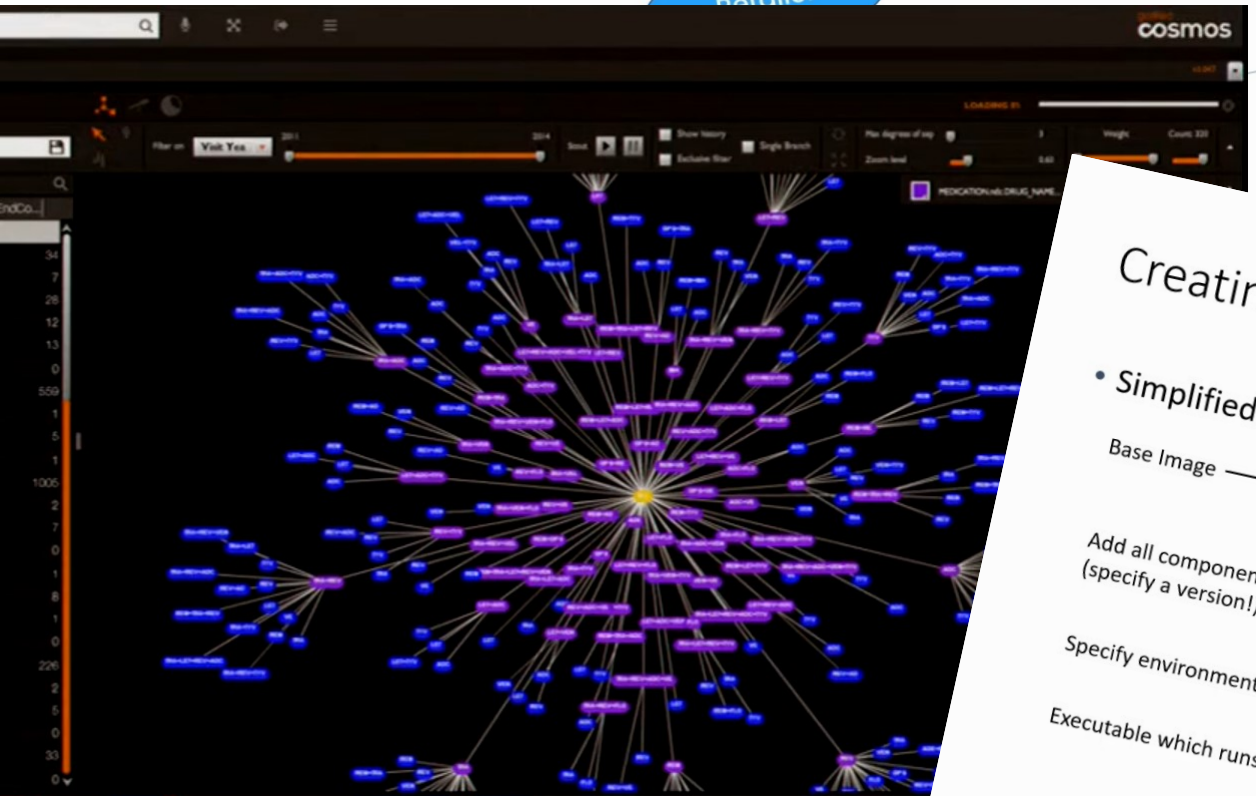


DYALOG

APL in Academia



APL in Industry



Creating a Custom Docker Image

- Simplified version of our custom Dockerfile

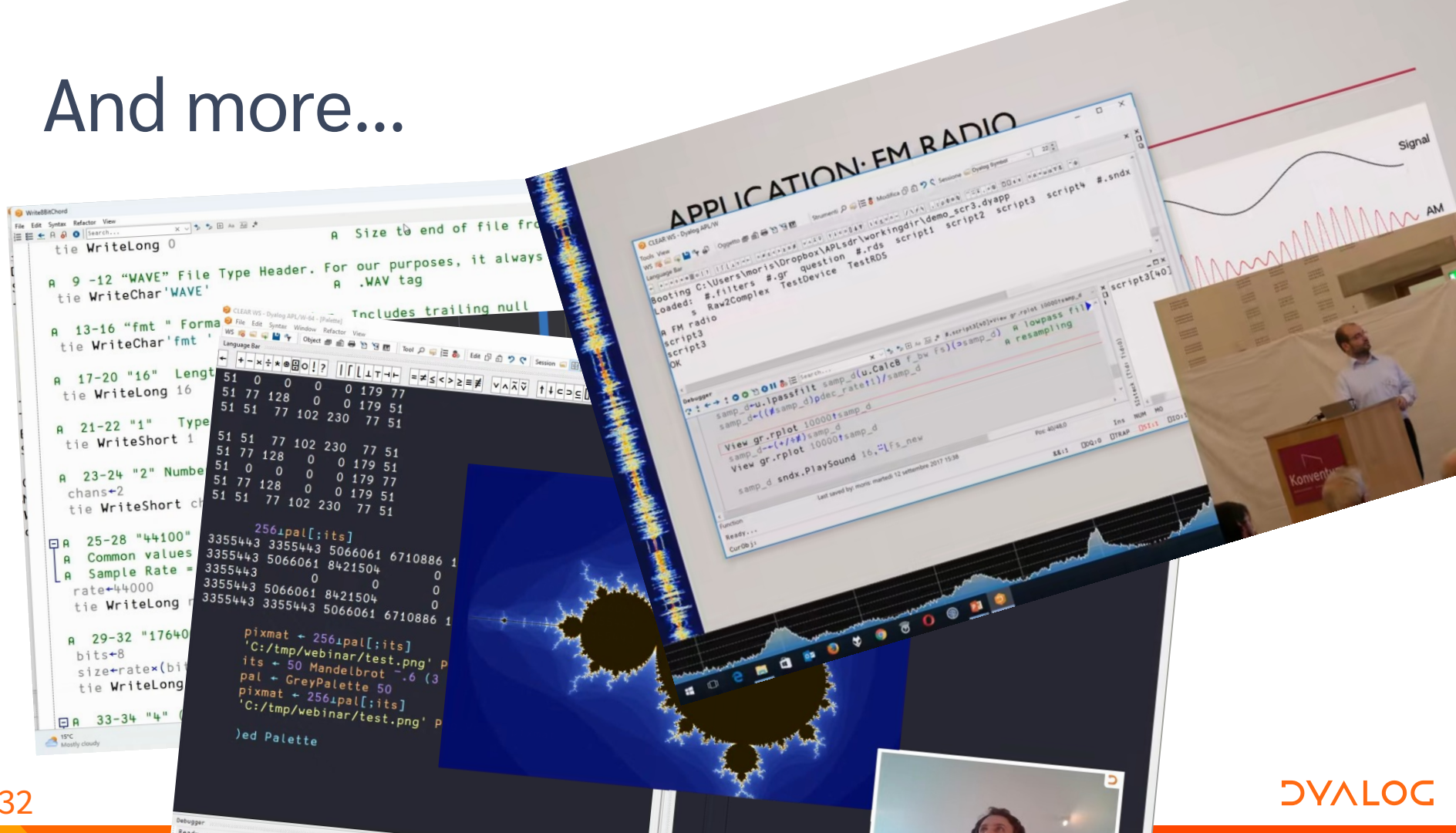
Base Image → `FROM redhat/ubi8-minimal:8.8`

Add all components (specify a version!) →
```
ADD APLSource /app
ADD linux_64_18.2.45405_unicode.x86_
RUN git clone https://github.com/dya
```

Specify environment variables →
```
ENV JarvisConfig="/app/Config.
ENV LOAD="/Jarvis/Source
```

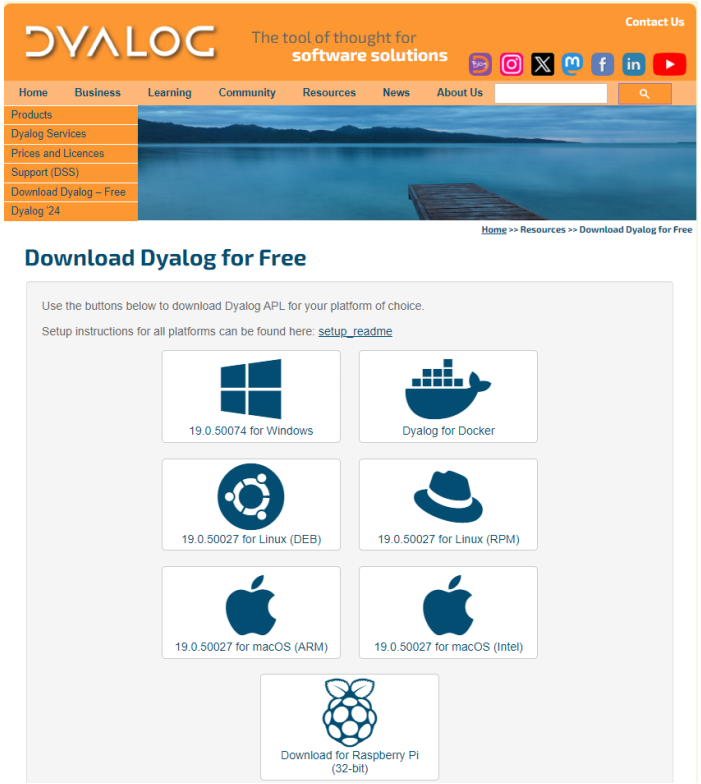
Executable which runs at startup →

And more...

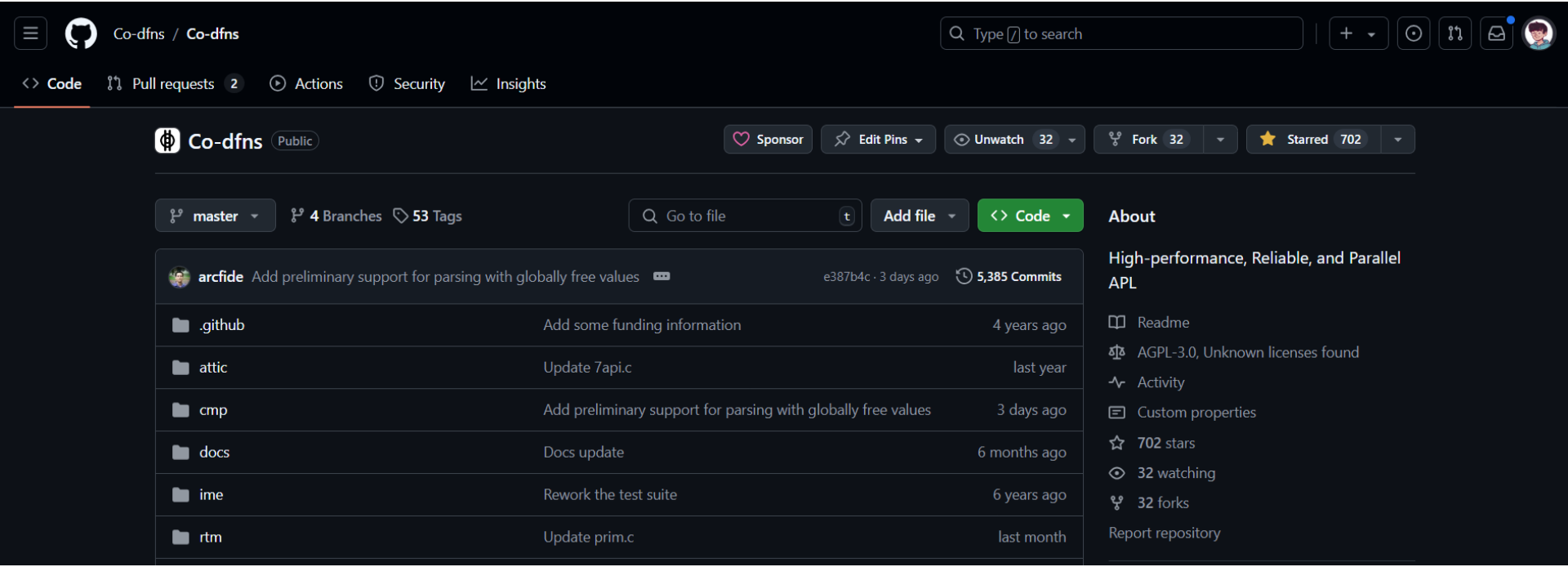


DYALOG

Dyalog Interpreter



Co-dfns Compiler



Everyone should, at least once

https://tryapl.org/

Dyalog APL

Dyalog IDE
For Windows

Dyalog IDE
For Unix

Tatin
Package Manager

Cider
Project Manager

- 6 platforms
- Object Oriented
- Functional

- UI Tools
- Interop-able
- Multilingual

Getting started

- Chat in the APL Orchard
- Ask a question on Stack Overflow (apl, dyalog)
- Use the r/apljk subreddit
- Learning APL by Stefan Kruger
- Checkout on YouTube:
  - @DyalogLtd
  - @DyalogUserMeetings

So, Dyalog APL

- Symbolic concise syntax
- Array Oriented Programming language
- Object oriented and functional programming
- High performance
- Cross platform

Notation as a Tool of Thought

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

Thank you!