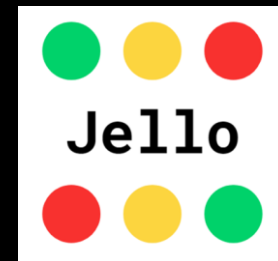


The Power of Function Composition



Conor Hoekstra

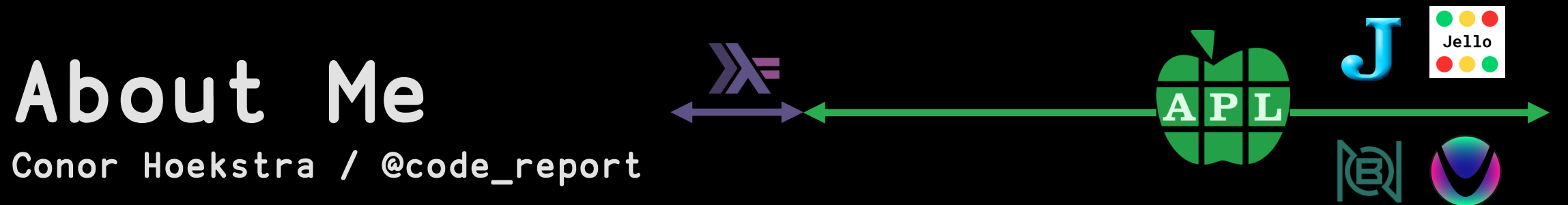
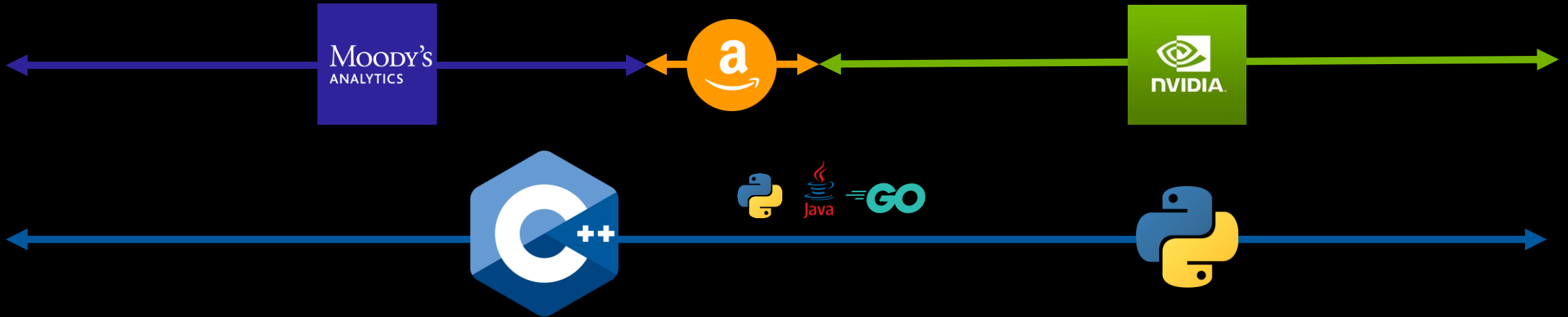
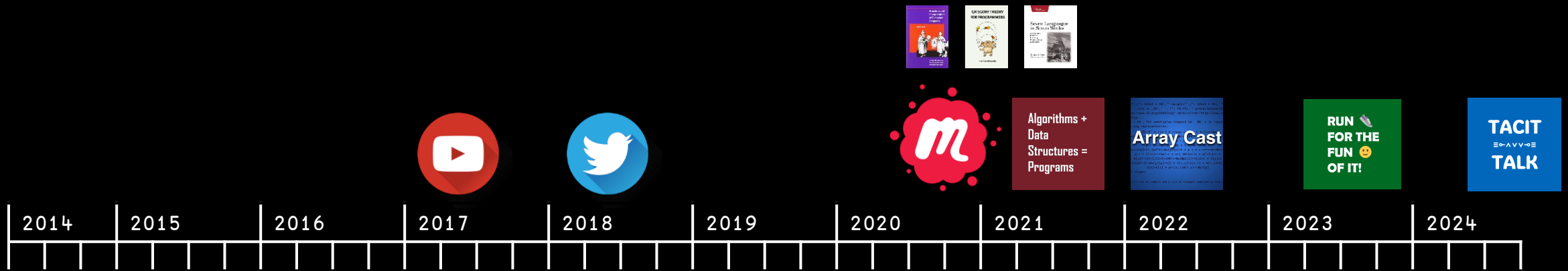


code_report



codereport





About Me
Conor Hoekstra / @code_report



344 Videos



39 (27) Talks

Algorithms + Data Structures = Programs

185 Episodes
@adspthepodcast



Array Cast

80 Episodes
@arraycast



TACIT
≡ ∅ ∧ ∨ ∨ ∅ ≡
TALK

2 Episodes
@codereport



RUN  FOR THE FUN  OF IT!

18 Episodes
@conorhoekstra

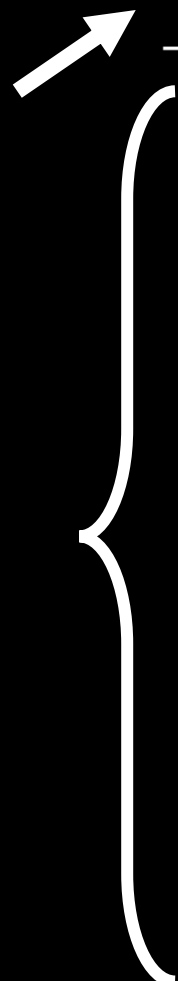


<https://github.com/codereport/Content>

Function Composition

Function Composition

1. Operators
2. Functions
3. Trains
4. Chains
5. Stacks*



FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⋄	⋄	~	~	`	.	<code>join</code>
C	⋄	⋄	~	~	@	:	<code>flip</code>
B	∘∘∘	∘	@:&:	∘O	*	*	.
B ₁	∘	∘	@:	∘	*	*	∴
S		∘	*	∘	*	*	<code>ap</code> / <*>
Σ		∘	*	∘	*	*	=<<
D	∘	∘	*	∘	*	*	
Δ		∘		∘	*	*	
Ψ	∘	∘	&:	O	*	∩	<code>on</code>
D ₂		a∘b∘c		a→b→c		⌈	
Φ	*	a<>c	*	*	*	⊃	<code>liftA2</code>
Φ ₁	*	a<>c	*	*	*	⊃	

Operators

Functions

Trains

Chains

Stacks*

combinator: a function that deals only in its arguments

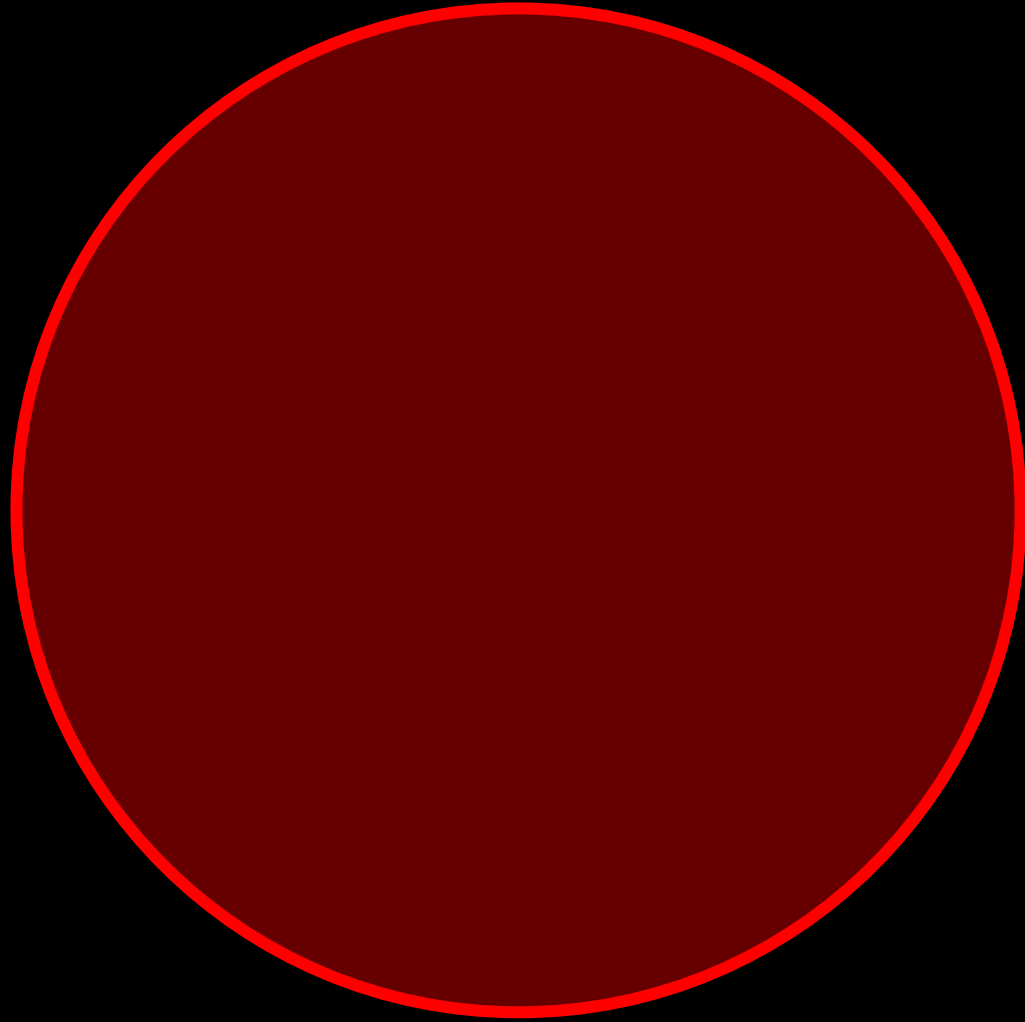
CL combinator: a combinator from Combinatory Logic

FOF: a combinator that only consumes **AND** produces functions

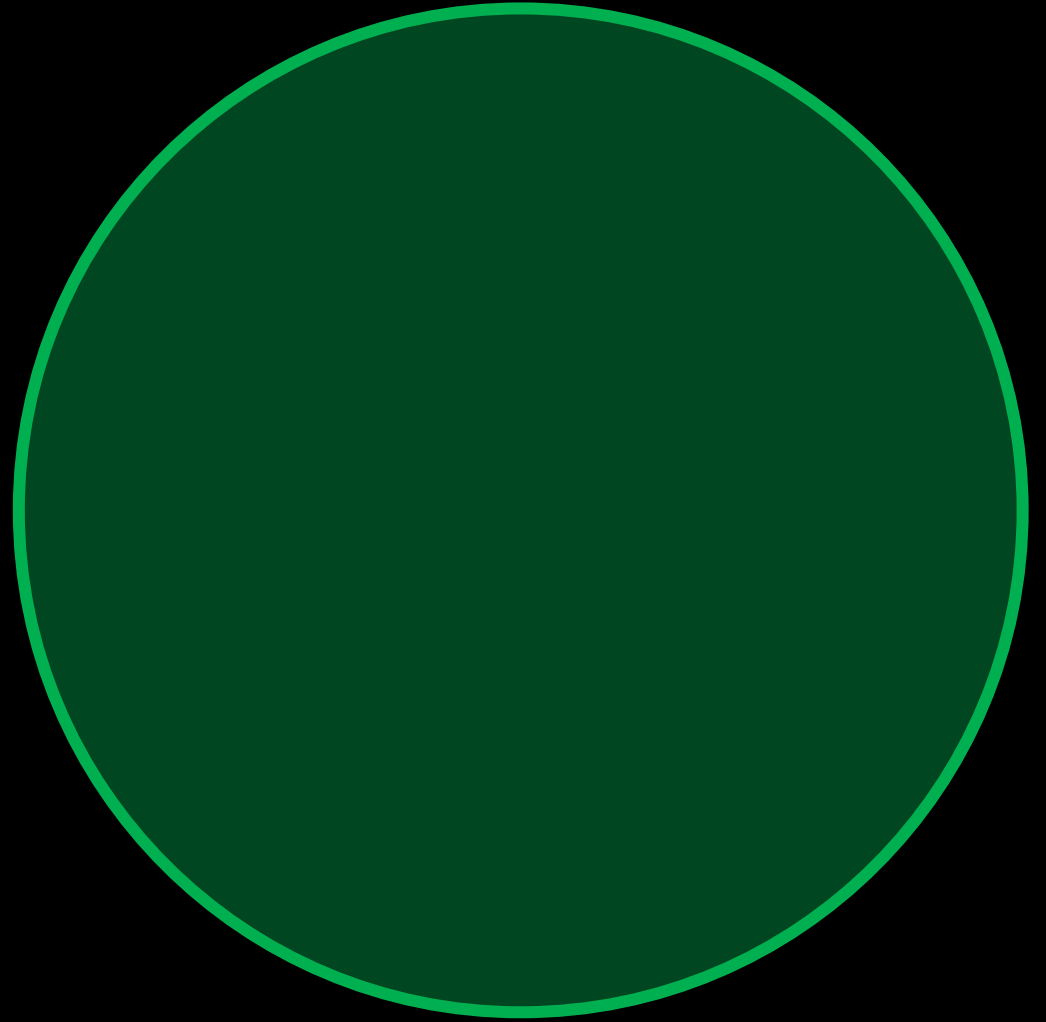
pure function: same input = same output / no side effects

HOF: consumes **OR** produces a function

pure function

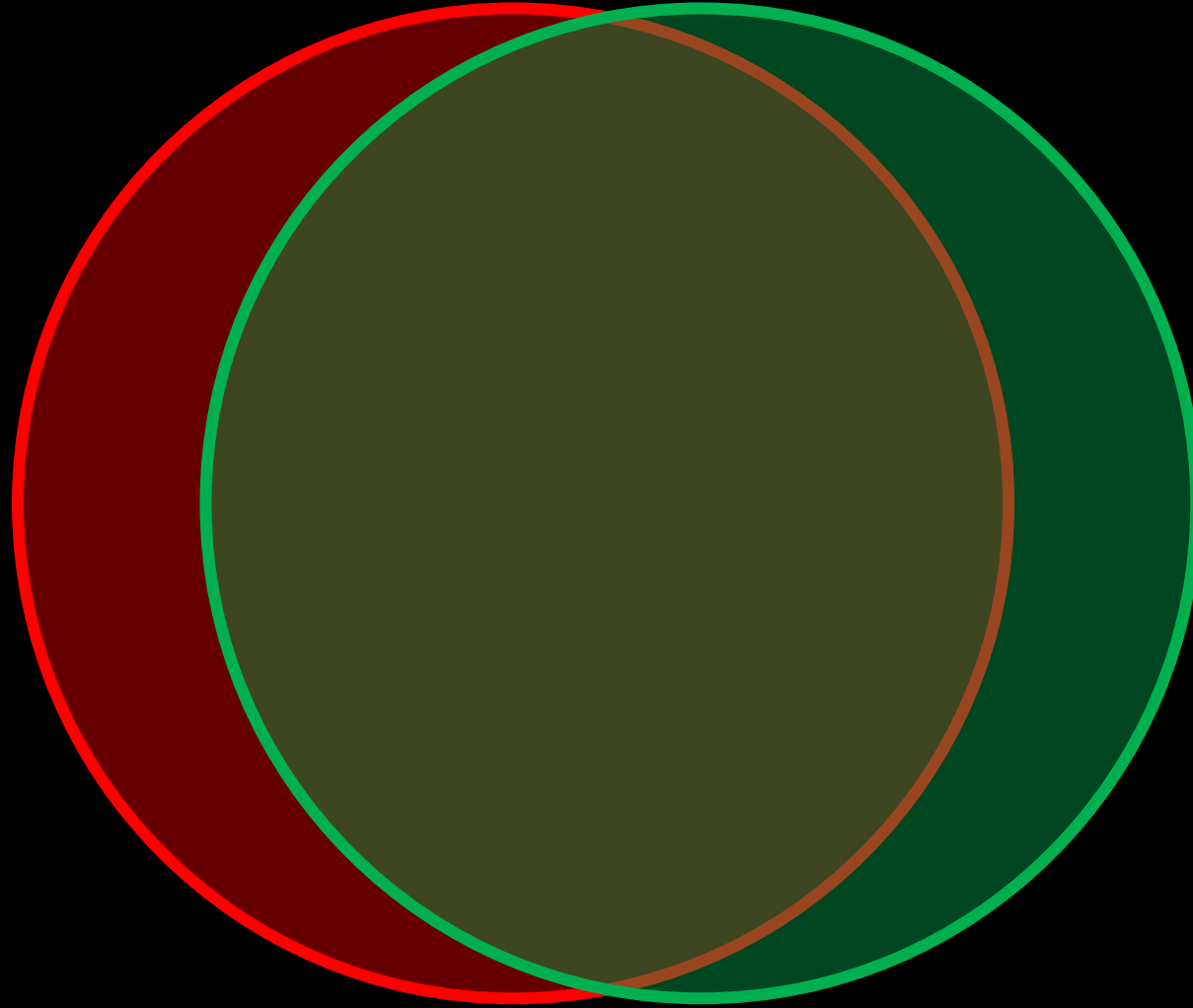


combinator



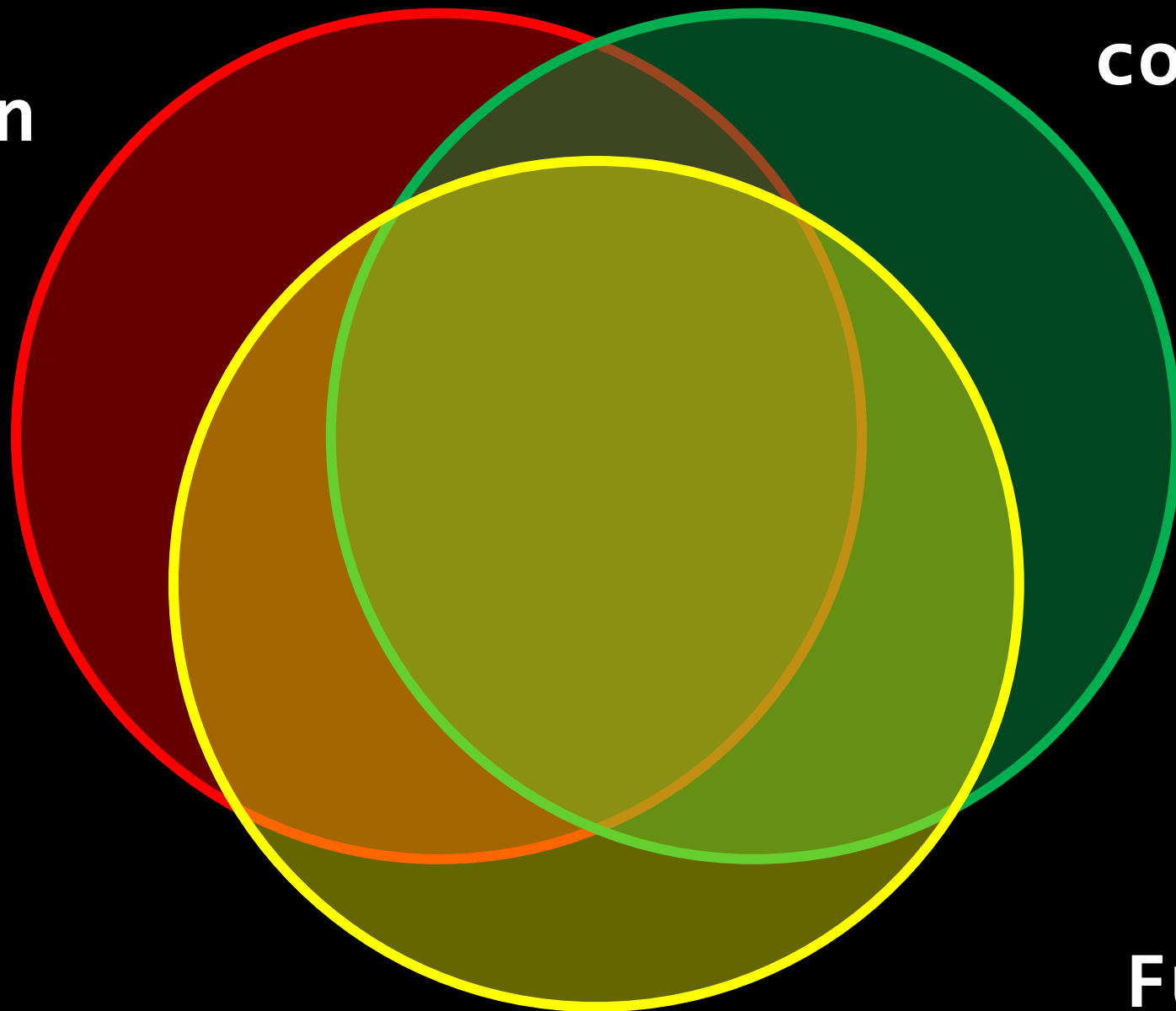
pure function

combinator



**pure
function**

combinator

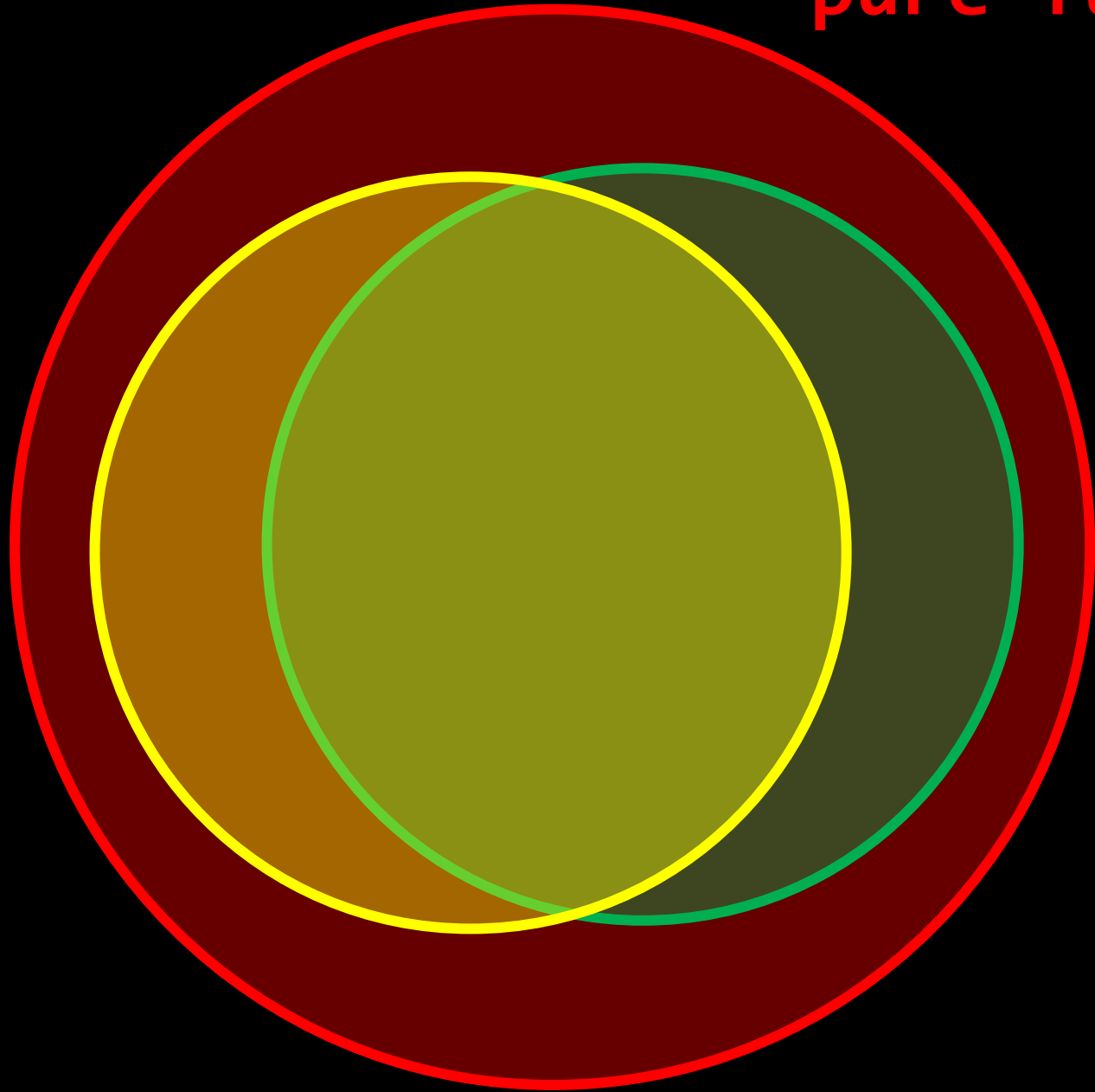


**HOF
(Higher
Order
Function)**

pure function

HOF

combinator



pure function

HOF

combinator

CL (SKI)
combinator



pure function

HOF

combinator

CL combinator

SBCW Φ Ψ

KI



pure function

HOF

combinator

CL combinator

SBCW Φ Ψ

KI

FOF

(Function Only
Function)



pure function

CL combinator

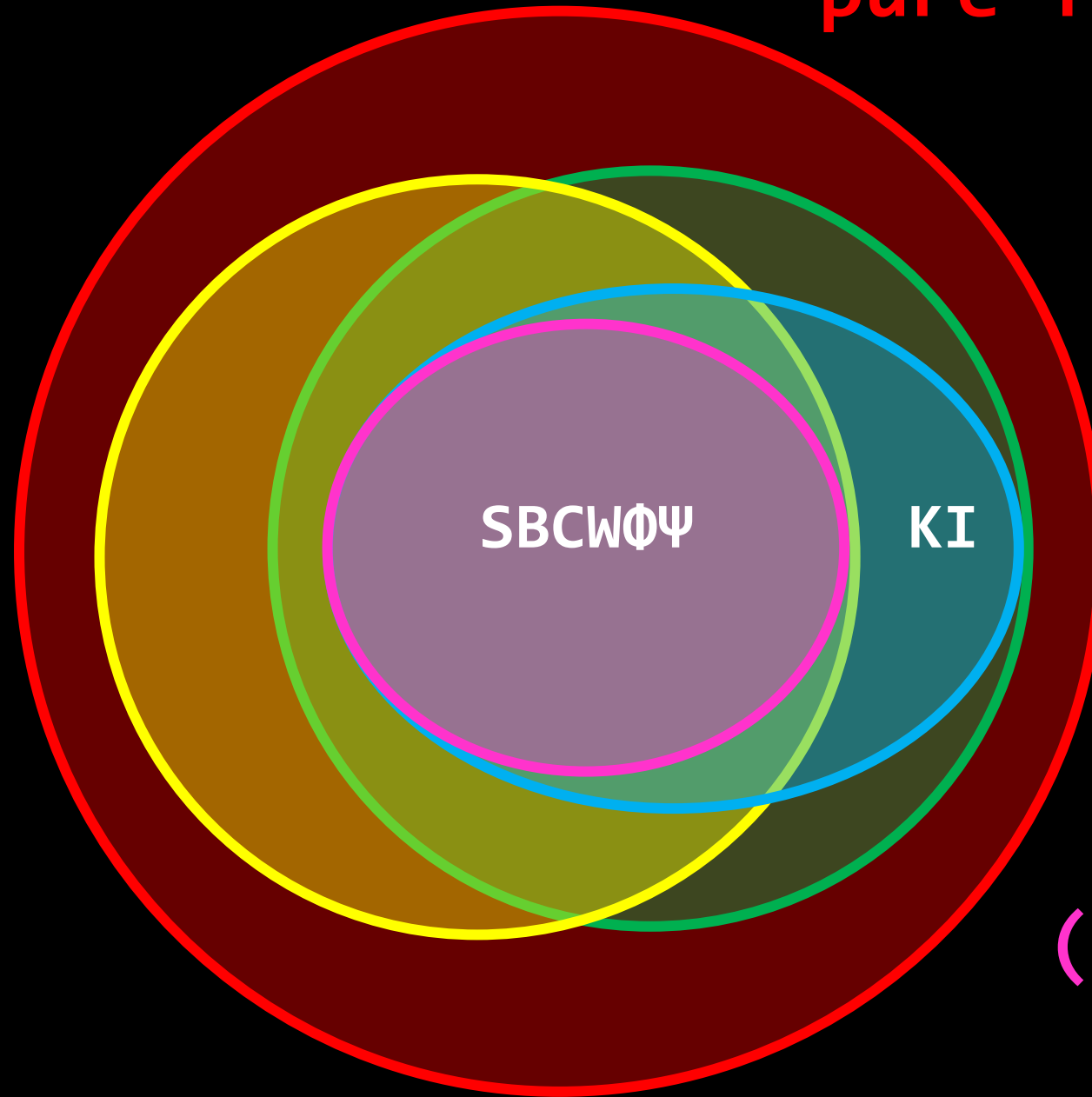
combinator

FOF
(Function Only
Function)

HOF

SBCW Φ Ψ

KI



combinator: a function that deals only in its arguments

CL combinator: a combinator from Combinatory Logic

FOF: a combinator that only consumes **AND** produces functions

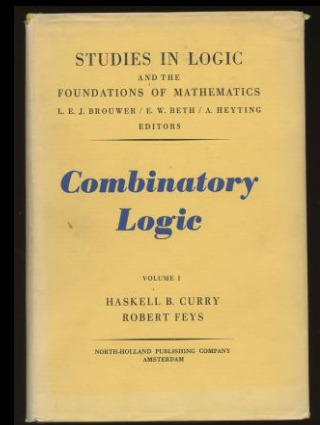
pure function: same input = same output / no side effects

HOF: consumes **OR** produces a function

Combinators

THE ELEMENTARY COMBINATORS

Combinator	Elementary Name
I	Elementary Identifier
C	Elementary Permutator
W	Elementary Duplicator
B	Elementary Compositor
K	Elementary Cancellator





```
def i(x):  
    return x
```



```
def k(x, y):  
    return x
```



```
def w(f):  
    return lambda x: f(x, x)
```

[[digression]]



Conor Hoekstra @code_report · Jan 8, 2022



Also, I apologize for my above average number of tweets 🐦 today, but this table of Greek/Latin words for describing function **arity** will be necessary for a future talk.

The \hat{E} combinator is "tetradic"

Unary/Monadic

Binary/Dyadic

Ternary/Triadic

Quaternary/Tetradic

Terminology [\[edit\]](#)

[Latinate](#) names are commonly used for specific arities, primarily based on [cardinal numbers](#) or [ordinal numbers](#). For example, 1-ary is based on

x-ary	Arity (Latin based)	Adicity (Greek based)
0-ary	<i>Nullary</i> (from <i>nūllus</i>)	<i>Niladic</i>
1-ary	<i>Unary</i>	<i>Monadic</i>
2-ary	<i>Binary</i>	<i>Dyadic</i>
3-ary	<i>Ternary</i>	<i>Triadic</i>
4-ary	<i>Quaternary</i>	<i>Tetradic</i>



6



2



46



[[end of digression]]



```
def w(f):  
    return lambda x: f(x, x)
```



```
def b(f, g):  
    return lambda x: f(g(x))
```



```
def c(f):  
    return lambda x, y: f(y, x)
```



```
def s(f, g):  
    return lambda x: f(x, g(x))
```



```
def i (x):      return x
def k (x, y):   return x
def ki (x, y):  return y
def s (f, g):   return lambda x:    f(x, g(x))
def b (f, g):   return lambda x:    f(g(x))
def c (f):      return lambda x, y: f(y, x)
def w (f):      return lambda x:    f(x, x)
def d (f, g):   return lambda x, y: f(x, g(y))
def b1 (f, g):  return lambda x, y: f(g(x, y))
def psi(f, g):  return lambda x, y: f(g(x), g(y))
def phi(f, g, h): return lambda x:  g(f(x), h(x))
```

Example

Example Special Array


<https://leetcode.com/problems/special-array-i/description/>

3151. Special Array I

Easy

 Topics

 Companies

 Hint

An array is considered **special** if every pair of its adjacent elements contains two numbers with different parity.

You are given an array of integers `nums`. Return `true` if `nums` is a **special** array, otherwise, return `false`.

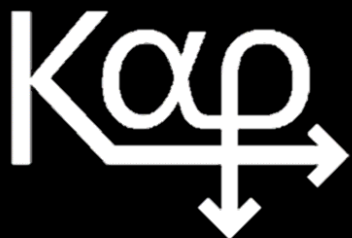
4 3 1 6

4	3	3	1	1	6
---	---	---	---	---	---

0	1	1	1	1	0
---	---	---	---	---	---

1 0 1

0





Python

Function Composition

- ✓ Functions
- ✗ Operators
- ✗ Trains
- ✗ Chains
- ✗ Stacks



```
def isArraySpecial(nums):  
    for i in range(len(nums) - 1):  
        if nums[i] % 2 == nums[i + 1] % 2:  
            return False  
    return True
```



```
def isArraySpecial(nums):  
    for x, y in zip(nums, nums[1:]):  
        if x % 2 == y % 2:  
            return False  
    return True
```



```
def isArraySpecial(nums):  
    return all(x % 2 != y % 2  
               for x, y in zip(nums, nums[1:]))
```



```
from dovekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(psi(ne, odd)(x, y)
               for x, y in zip(nums, nums[1:]))
```



```
def psi(f, g):  
    return lambda x, y: f(g(x), g(y))
```



```
from doviekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(psi(ne, odd)(x, y)
               for x, y in zip(nums, nums[1:]))
```



```
from doviekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```




Haskell

Function Composition

- ✓ Functions
- ✓ Operators
- ✗ Trains
- ✗ Chains
- ✗ Stacks



```
isArraySpecial = foldl1 (&&  
    . ((zipWith (/=)) <*> tail)  
    . map odd
```

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⋄	⋄	~	~	`	.	<code>join</code>
C	⋄	⋄	~	~	@	:	<code>flip</code>
B	∘∘∘	∘	@:&:	∘O	*	*	.
B ₁	∘	∘	@:	∘	*	*	∴
S		∘	*	∘	*	*	<code>ap</code> / <*>
Σ		∘	*	∘	*	*	=<<
D	∘	∘	*	∘	*	*	
Δ		∘		∘	*	*	
Ψ	∘	∘	&:	O	*	∩	<code>on</code>
D ₂		a∘b∘c		a→b→c	*	⊐	
Φ	*	a<>c	*	*	*	⊃	<code>liftA2</code>
Φ ₁	*	a<>c	*	*	*	⊃	

Operators

Functions

Trains

Chains

Stacks*



```
isArraySpecial = foldl1 (&&  
    . ((zipWith (/=)) <*> tail)  
    . map odd
```



```
import Data.List.HT (mapAdjacent)

isArraySpecial = foldl1 (&&)
                  . mapAdjacent (/=)
                  . map odd
```



```
import Data.List.HT (mapAdjacent)
```

```
isArraySpecial = and  
    . mapAdjacent (/=)  
    . map odd
```



```
import Data.List.HT (mapAdjacent)
import Data.Function (on)

isArraySpecial = and
                . mapAdjacent (on (/=) odd)
```



```
from dovekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```




```
from doviekie import psi, odd  
from operator import ne
```

```
def isArraySpecial(nums):  
    return all(map(psi(ne, odd), nums, nums[1:]))
```



```
from dovekier import psi, odd
from operator import ne

def adjacentMap(xs, op):
    return [op(a, b) for a, b in pairwise(xs)]

def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```



```
from dovekier import psi, odd
from operator import ne

def adjacentMap(xs, op):
    return [op(a, b) for a, b in pairwise(xs)]

def isArraySpecial(nums):
    return all(adjacentMap(nums, psi(ne, odd)))
```



```
from dovekie import odd
from operator import ne

def adjacentMap(xs, op):
    return [op(a, b) for a, b in pairwise(xs)]

def isArraySpecial(nums):
    return all(adjacentMap(map(odd, nums), ne))
```



Clojure

Function Composition

- ✓ Functions
- ✓ Operators
- ✗ Trains
- ✗ Chains
- ✗ Stacks



```
(defn is-special-array [nums]
  (->> nums
    (partition 2 1)
    (map #(reduce not= %))
    (every? identity)))
```



```
(defn is-special-array [nums]
  (->> nums
    (partition 2 1)
    (every? (fn [[a b]] not= a b))))
```



APL, BQN, J, Kap

Function Composition

✗ Functions

✓ Operators

✓ Trains

✗ Chains

✗ Stacks

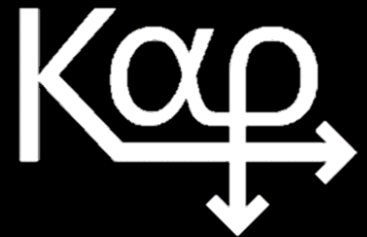


Table 4: 2 and 3-trains in APL, Kap, BQN and J.

Year	Language	2-Train	3-Train
1990	J	S and D	Φ and Φ_1
2014	Dyalog APL	B and B_1	Φ and Φ_1
2020	Kap	B and B_1	-
2020	BQN	B and B_1	Φ and Φ_1



```
def i (x):      return x
def k (x, y):   return x
def ki (x, y):  return y
def s (f, g):   return lambda x:      f(x, g(x))
def b (f, g):   return lambda x:      f(g(x))
def c (f):      return lambda x, y:  f(y, x)
def w (f):      return lambda x:      f(x, x)
def d (f, g):   return lambda x, y:  f(x, g(y))
def b1 (f, g):  return lambda x, y:  f(g(x, y))
def psi(f, g):  return lambda x, y:  f(g(x), g(y))
def phi(f, g, h): return lambda x:    g(f(x), h(x))
```



```
def s (f, g):      return lambda x:      f(x, g(x))  
def b (f, g):      return lambda x:      f(g(x))
```

```
def d (f, g):      return lambda x, y: f(x, g(y))  
def b1 (f, g):      return lambda x, y: f(g(x, y))
```

```
def phi(f, g, h):  return lambda x:      g(f(x), h(x))
```

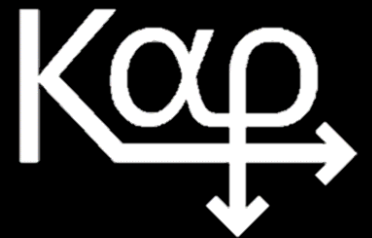


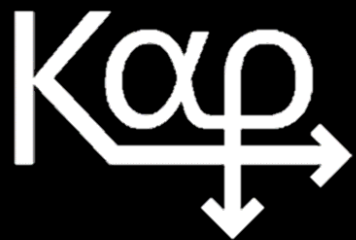
```
def b (f, g): return lambda x: f(g(x))
def b1 (f, g): return lambda x, y: f(g(x, y))

def phi (f, g, h): return lambda x: g(f(x), h(x))
def phi1(f, g, h): return lambda x, y: g(f(x, y), h(x, y))
```

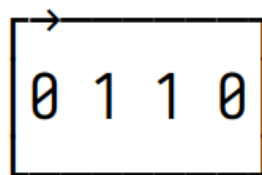


Live Coding

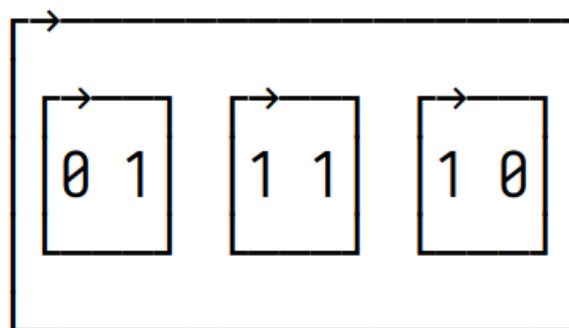




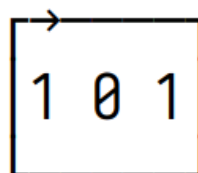
2|4 3 1 6



2,/2|4 3 1 6



2≠/2|4 3 1 6



Λ/2≠/2|4 3 1 6

0

(Λ/2≠/2|)4 3 1 6

0

Jelly

Function Composition






 Functions

 Operators

 Trains

 Chains

 Stacks

   Jello   

> [4,1,6]

●●● Jello ●●●

> [4,1,6] :: odd?

●

●

[4,1,6]



[0, 1, 0]

●●● Jello ●●●

> [4,1,6] :: odd? differ

Ĥ

Ď

Ĥ [4,1,6] → [0, 1, 0]

ĤĎ [4,1,6] → [1, 1]

●●● Jello ●●●

> [4,1,6] :: odd? differ all

	\dot{B}		\check{D}		\dot{A}
\dot{B}	[4,1,6]	→	[0, 1, 0]		
$\dot{B}\check{D}$	[4,1,6]	→	[1, 1]		
$\dot{B}\check{D}\dot{A}$	[4,1,6]	→	1		

●●● Jello ●●●

> [4,1,6] :: odd? differ all

	\dot{B}		\check{D}		\dot{A}
\dot{B}	[4,1,6]	→		[0, 1, 0]	
$\dot{B}\check{D}$	[4,1,6]	→		[1, 1]	
$\dot{B}\check{D}\dot{A}$	[4,1,6]	→		1	

This is a 1-1-1 monadic chain (BB)



●●● Jello ●●●

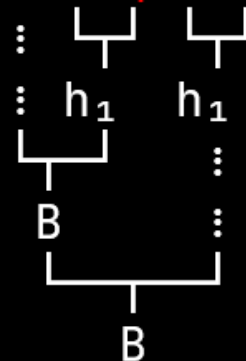
> [4,1,6] :: odd? ≠ prior and fold
 Ĥ n p a /

and fold can be replaced with all
👉🤖 algorithm advisor 🤖👉

≠ prior can be replaced with differ
👉🤖 algorithm advisor 🤖👉

Ĥ [4,1,6] → [0, 1, 0]
Ĥn [4,1,6] → [1, 1]
Ĥnpa/ [4,1,6] → 1

This is a 1-2-q-2-q monadic chain (BB)





Ulua

Function Composition

✗ Functions

✓ Operators

✗ Trains

✗ Chains

✓ Stacks



Kap

$\wedge / 2 \neq / 2 |$

B B



Dyalog APL

$\wedge / 2 \neq / 2 | \vdash$

B ϕ ϕ



Uiua

$/ \downarrow \equiv / \neq \boxtimes 2 \triangle 2$



BQN

$\wedge ' \cdot \neq ' \cup 2 \updownarrow 2 | \vdash$



B B ϕ ϕ



J

$[: * . / 2 \sim : / \backslash 2 | [$

B ϕ ϕ



Jello

odd? differ all

B B

In Conclusion



Kap

$\wedge/2\neq/2|$



Dyalog APL

$\wedge/2\neq/2|\vdash$



Uiua

$/\Downarrow\equiv/\neq\boxplus 2\triangle 2$



BQN

$\wedge' \cdot \neq' \cup 2\updownarrow 2|\vdash$



J

$[: * . / 2 \sim : / \backslash 2 | [$



Jello

odd? differ all

lucid, systematic,
and penetrating
treatment of basic
and dynamic data
structures, sorting,
recursive algorithms,
language structures,
and compiling

NIKLAUS WIRTH

Algorithms +
Data
Structures –
Programs

PRENTICE-HALL
SERIES IN
AUTOMATIC
COMPUTATION

Combinators =

Beautiful Code

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⋄	⋄	~	~	`	.	<code>join</code>
C	⋄	⋄	~	~	@	:	<code>flip</code>
B	∘∘∘	∘	@:&:	∘O	*	*	.
B ₁	∘	∘	@:	∘	*	*	∴
S		∘	*	∘	*	*	<code>ap</code> / <*>
Σ		∘	*	∘	*	*	=<<
D	∘	∘	*	∘	*	*	
Δ		∘		∘	*	*	
Ψ	∘	∘	&:	O	*	∩	<code>on</code>
D ₂		a∘b∘c		a→b→c	*	⊐	
Φ	*	a<>c	*	*	*	⊃	<code>liftA2</code>
Φ ₁	*	a<>c	*	*	*	⊃	

Operators

Functions

Trains

Chains

Stacks*



```
from doviekie import odd
from operator import ne

def isArraySpecial(nums):
    return all(adjacentMap(map(odd, nums), ne))
```



```
from dovekie import odd

def isArraySpecial(nums):
    return all(differ(map(odd, nums)))
```



```
from dovekie import odd

def isArraySpecial(nums):
    return all(differ(map(odd, nums)))
```

↑ ↖ ↗
reduce map



Kap

$\wedge/2\neq/2|$



Dyalog APL

$\wedge/2\neq/2|\vdash$



Uiua

$/\Downarrow\equiv/\neq\boxplus 2\Delta 2$



BQN

$\wedge' \cdot \neq' \cup 2\updownarrow 2|\vdash$



J

$[: * . / 2 \sim : / \backslash 2 | [$



Jello

odd? differ all

Online REPLS

Language	Link
APL	https://tryapl.org/
Kap	https://kapdemo.dhsdevelopments.com/clientweb2/
BQN	https://bqnpad.mechanize.systems/
J	https://jsoftware.github.io/j-playground/bin/html2/
Uiua	https://www.uiua.dev/pad



Dovekie



Blackbird



Dovekie

dovekie 0.7.0

`pip install dovekie` 



Blackbird

```
# --- Fetch blackbird -----  
  
FetchContent_Declare(blackbird  
  GIT_REPOSITORY https://github.com/codereport/blackbird  
  GIT_TAG main  
)  
  
FetchContent_GetProperties(blackbird)  
if(NOT blackbird_POPULATED)  
  FetchContent_Populate(blackbird)  
  add_subdirectory(${blackbird_SOURCE_DIR} ${blackbird_BINARY_DIR} EXCLUDE_FROM_ALL)  
endif()
```


www.combinatorylogic.com


Thank You



<https://github.com/codereport/Content/Talks>

Conor Hoekstra

 code_report


 codereport


Questions?



<https://github.com/codereport/Content/Talks>

Conor Hoekstra

 code_report

 codereport