

# The Power of Function Composition



Conor Hoekstra

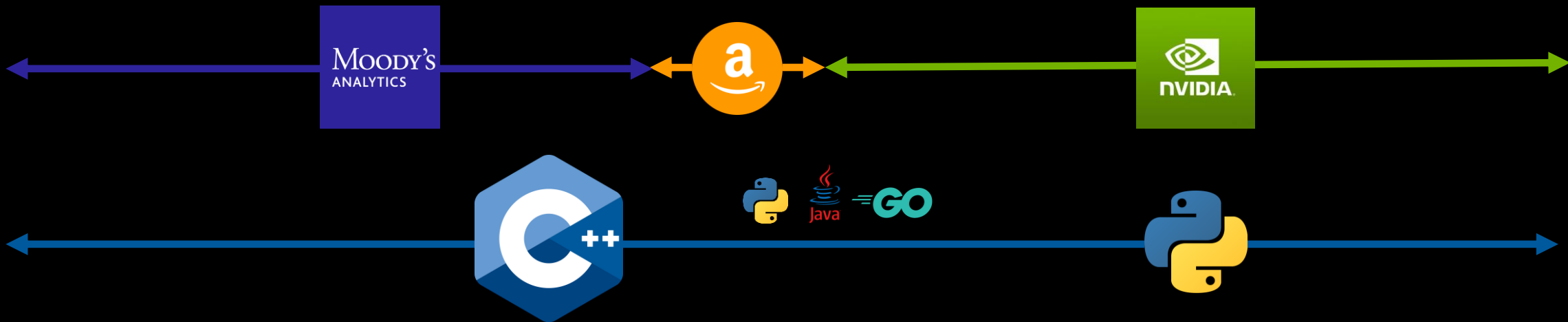
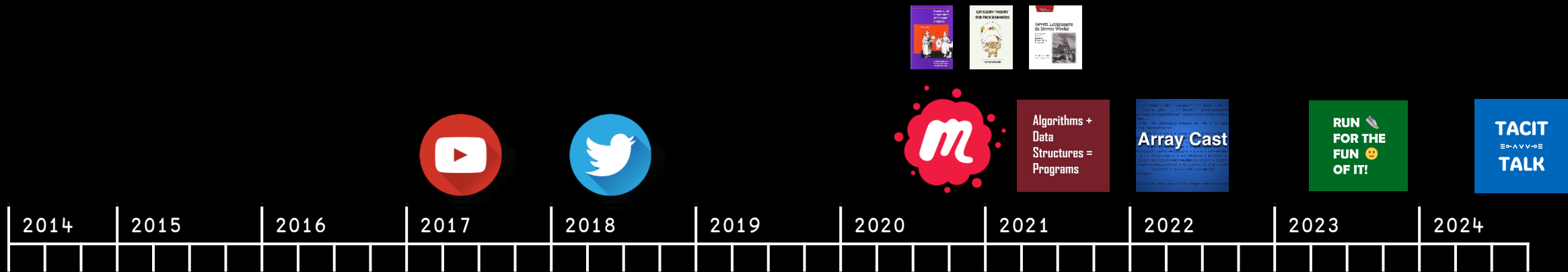


code\_report



codereport





# About Me

Conor Hoekstra / @code\_report





## 351 Videos



## 40 (28) Talks

# Algorithms + Data Structures = Programs

202 Episodes  
@adspthepodcast



# Array Cast

89 Episodes  
@arraycast



**TACIT**  
≡ ∅ ∨ ∨ ∅ ≡  
**TALK**

5 Episodes  
@codereport



**RUN **  
**FOR THE**  
**FUN **  
**OF IT!**

20 Episodes  
@conorhoekstra

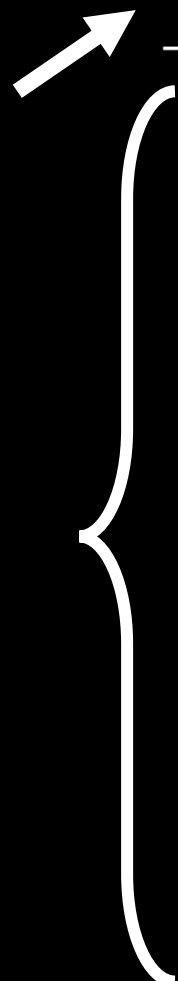


<https://github.com/codereport/Content>

# 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¨¨	¨	@:&:	oO	*	*	.
B <sub>1</sub>	¨	*	@:	o	*	*	::
S		o	*	⌘	*	*	<code>ap</code> / <*>
Σ		o		⌘	*	*	=<<
D	o	o	*	⌘	*	*	
Δ		o		⌘	*	*	
Ψ	¨	¨	&:	O	*	∩	<code>on</code>
D <sub>2</sub>		a <u>o</u> b◊c		a◊b◊c	*	⌈	
Φ	*	a<<b>>c	*	*	*	⌋	<code>liftA2</code>
Φ <sub>1</sub>	*	a<<b>>c	*	*	*	⌋	

Operators

Functions

Trains

Chains

Stacks\*

**combinator:** a function that deals only in its arguments

**CL combinator:** a combinator from Combinatory Logic

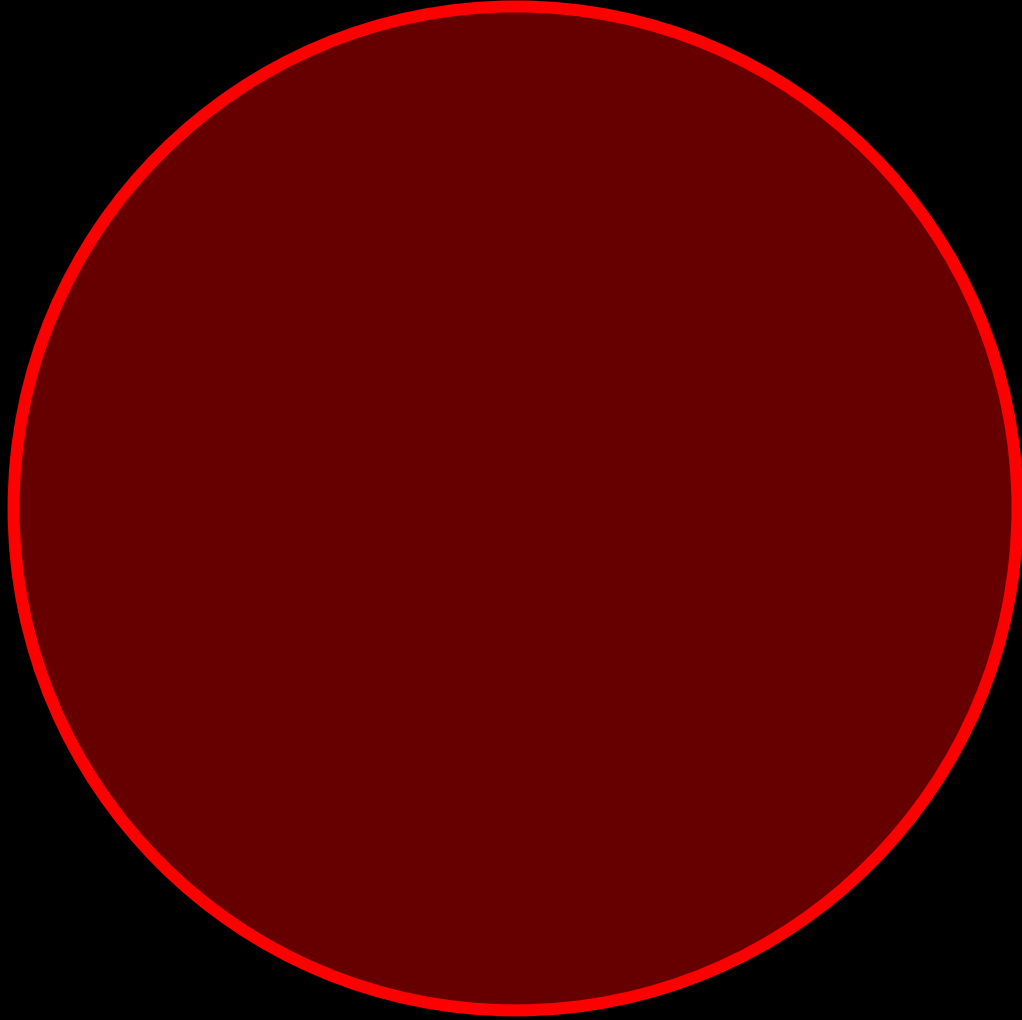
**FOF:** a combinator that only consumes **AND** produces functions

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

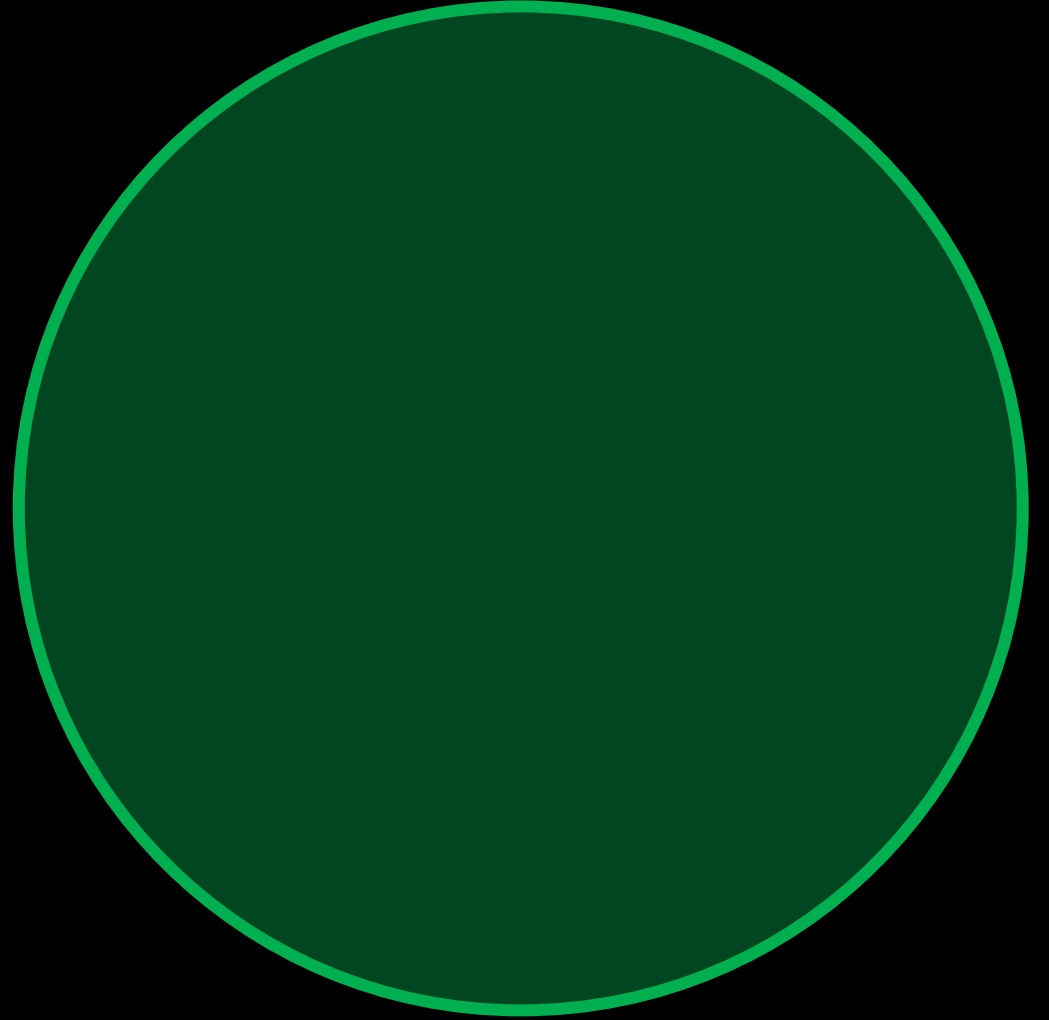
**HOF:** consumes **OR** produces a function



**pure function**

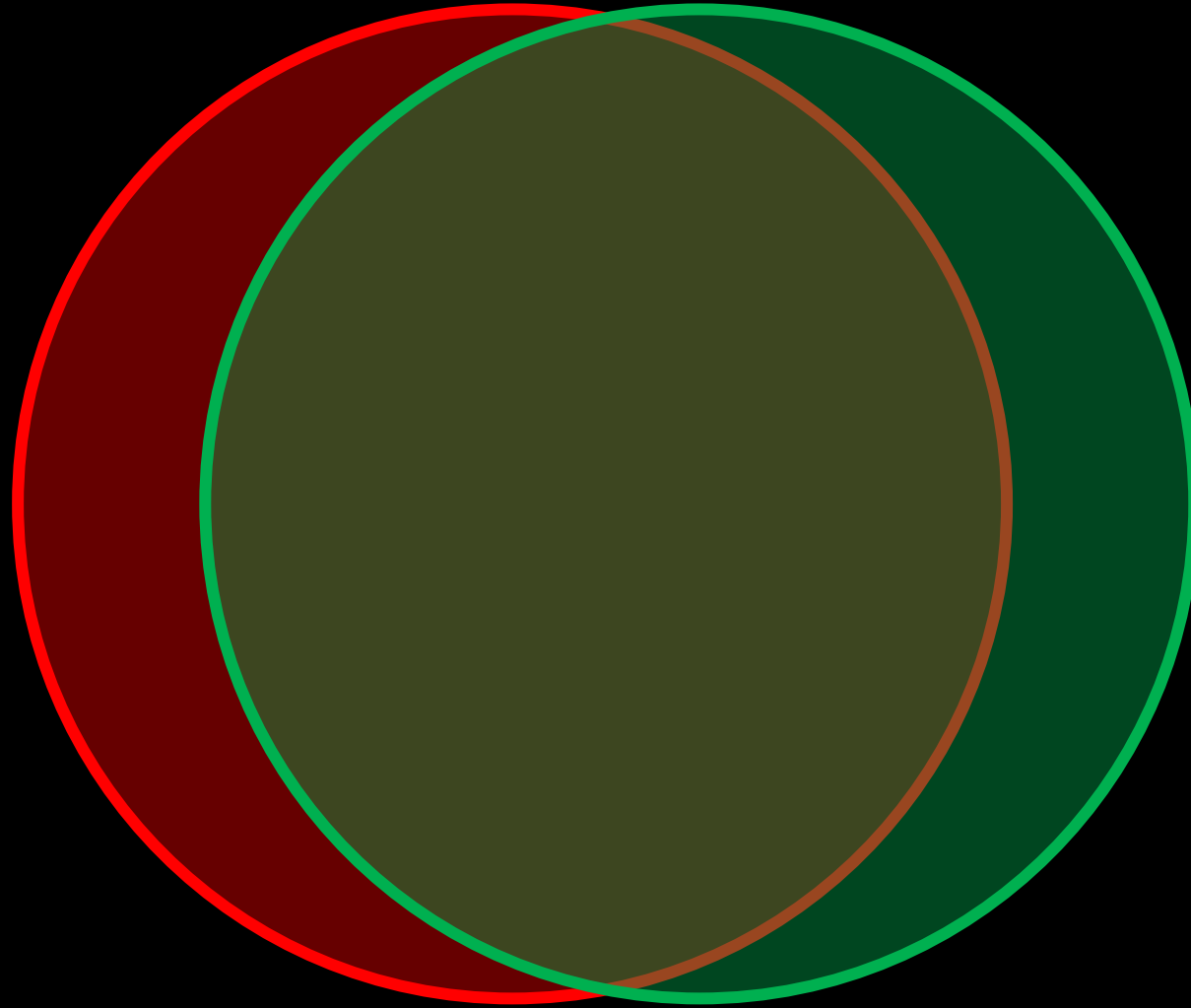


**combinator**



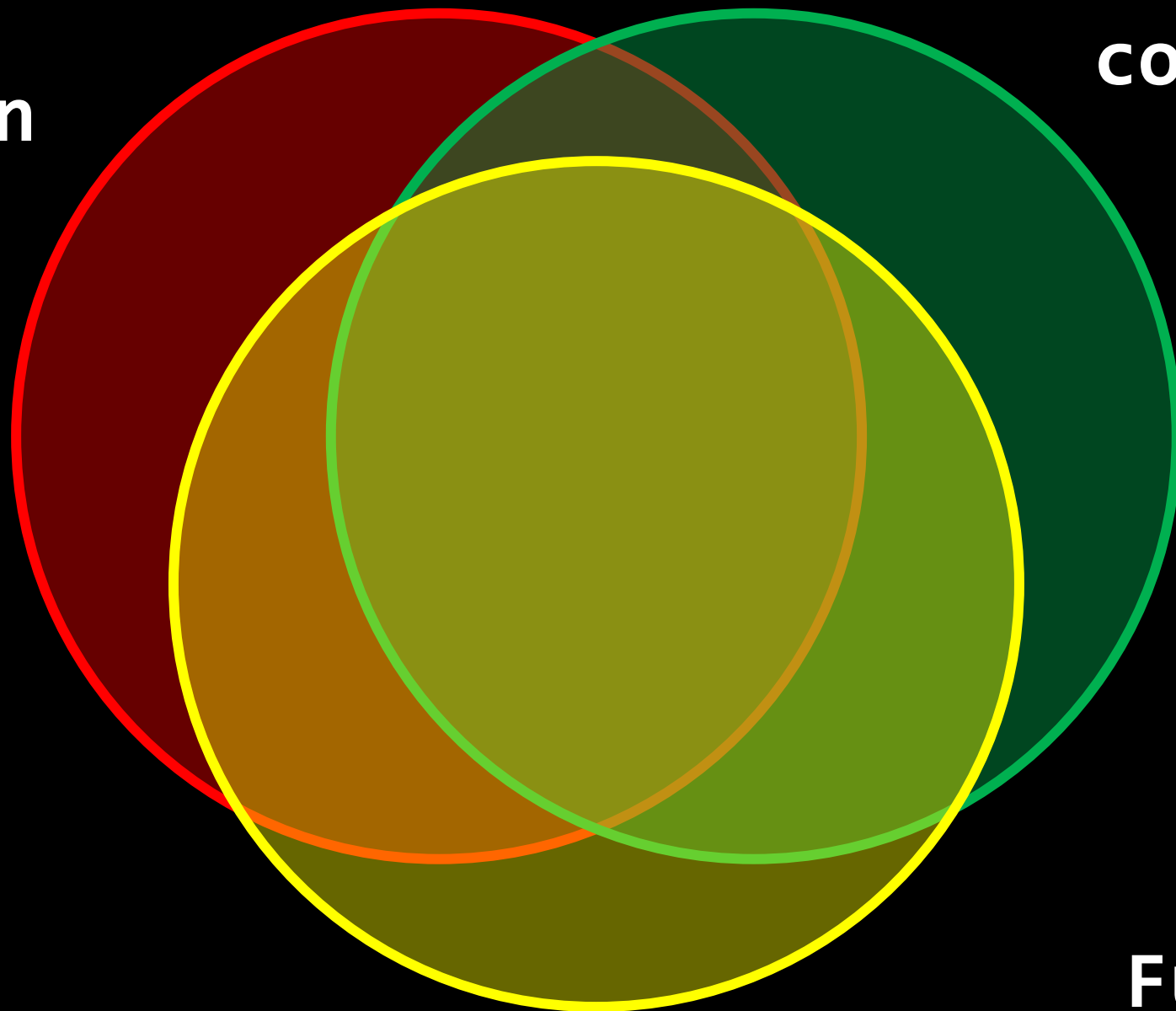
pure function

combinator



**pure  
function**

**combinator**

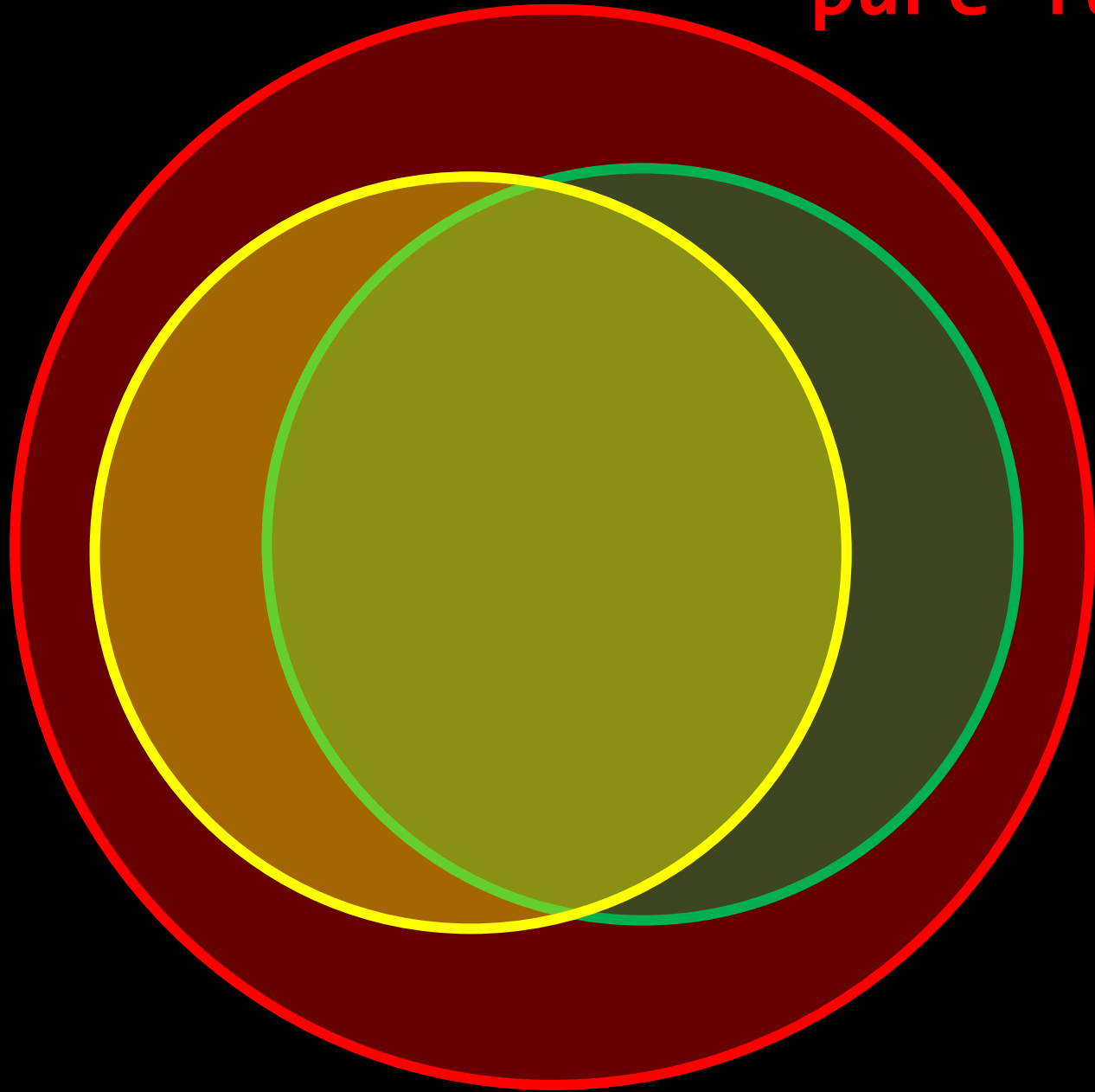


**HOF  
(Higher  
Order  
Function)**

pure function

HOF

combinator



pure function

HOF

combinator

CL (SKI)  
combinator



pure function

HOF

combinator

CL combinator

SBCW $\Phi$  $\Psi$

KI



pure function

HOF

combinator

CL combinator

SBCW $\Phi$  $\Psi$

KI

FOF

(Function Only  
Function)



pure function

CL combinator

combinator

FOF  
(Function Only  
Function)

HOF

SBCW $\Phi$  $\Psi$

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

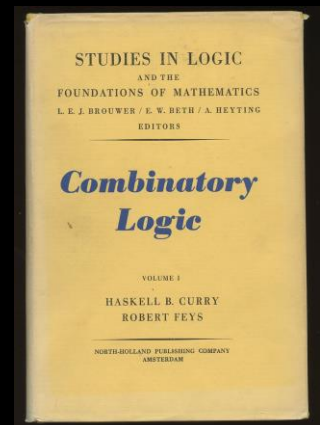
**pure function:** same input = same output / no side effects

**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



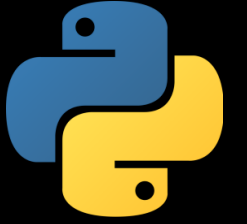
46



**[[ end of digression ]]**



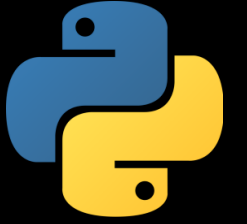
```
def w(f):  
    return lambda x: f(x, x)
```



```
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
---	---	---	---	---	---

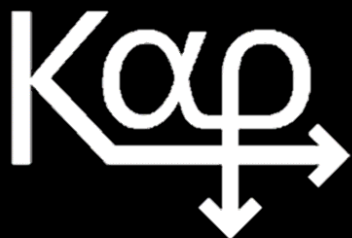
0	1	1	1	1	0
---	---	---	---	---	---

$0 \neq 1$	$1 \neq 1$	$1 \neq 0$
------------	------------	------------

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:]))
```



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



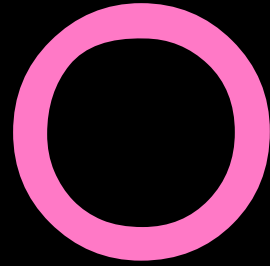
```
from doviekie import psi, odd
from operator import ne

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

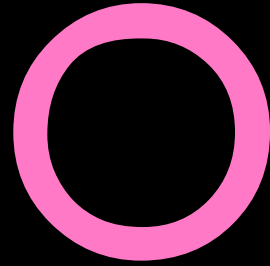


```
def psi(f, g):  
    return lambda x, y: f(g(x), g(y))
```





```
_Psi_ ← { (Gw) F Gx }  
      # Over  
      # on (Haskell)
```



```
_Psi_ ← { (Gw) F (Gx) }  
      # Over  
      # on (Haskell)
```



```
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:]))
```



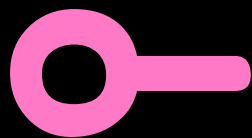
# Haskell

## Function Composition

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



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



S  $\leftarrow \{x F(Gx)\}$   
# After (Monadic)  
# Hook (J/I)

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	¨	¨	~	~	`	.	<code>join</code>
C	¨	¨	~	~	@	:	<code>flip</code>
B	o¨¨	¨	@:&:	oO	*	*	.
B <sub>1</sub>	¨	*	@:	o	*	*	::
S		o	*	o	*	*	<code>ap</code> / <*>
Σ		o		o	*	*	=<<
D	o	o	*	o	*	*	
Δ		o		o	*	*	
Ψ	¨	¨	&:	O	*	∩	<code>on</code>
D <sub>2</sub>		a <u>o</u> b◦c		a◊b◊c		⊐	
Φ	*	a<<b>>c	*	*	*	⊃	<code>liftA2</code>
Φ <sub>1</sub>	*	a<<b>>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 doviekie import psi, odd
from operator import ne

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



```
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
from itertools import pairwise

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 doviekie import psi, odd
from operator import ne
from itertools import pairwise

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 doviekie import odd
from operator import ne
from itertools import pairwise

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

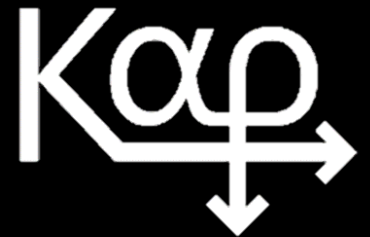
✗ Functions

✓ Operators

✓ Trains

✗ Chains

✗ Stacks



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

<b>Year</b>	<b>Language</b>	<b>2-Train</b>	<b>3-Train</b>
1990	J	S and D	$\Phi$ and $\Phi_1$
2014	Dyalog APL	B and $B_1$	$\Phi$ and $\Phi_1$
2020	Kap	B and $B_1$	-
2020	BQN	B and $B_1$	$\Phi$ and $\Phi_1$



```
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))
```



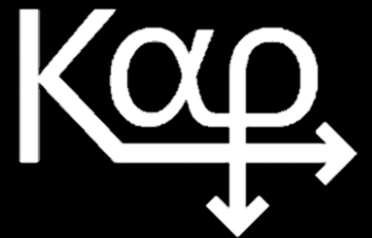


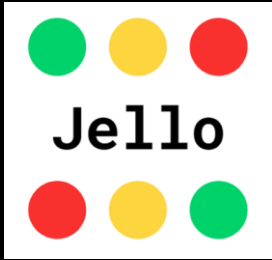
```
def b (f, g): return lambda x: f(g(x))
def b1 (f, g): return lambda x, y: f(g(x, y))

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





# Jelly

## Function Composition

✗ Functions

✓ Operators

✗ Trains

✓ Chains

✗ Stacks



# Uiua

## Function Composition

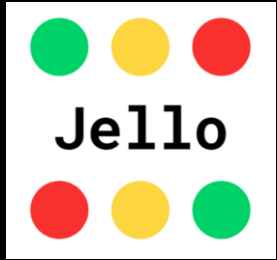
✗ Functions

✓ Operators

✗ Trains

✗ Chains

✓ Stacks



# Live Coding



Kap

$\wedge / 2 \neq / 2 \mid$

B B



Dyalog APL

$\wedge / 2 \neq / 2 \mid \vdash$

B  $\phi$   $\phi$



Uiua

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



BQN

$\wedge \cdot \neq \cdot \cup 2 \updownarrow 2 \mid \vdash$



B B  $\phi$   $\phi$



J

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

B  $\phi$   $\phi$



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

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



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¨¨	¨	@:&:	oO	*	*	.
B <sub>1</sub>	¨	*	@:	o	*	*	..
S		o	*	o		*	<code>ap</code> / <*>
Σ		o		o	*	*	=<<
D	o	o	*	o		*	
Δ		o		o	*	*	
Ψ	¨	¨	&:	O	*	∩	<code>on</code>
D <sub>2</sub>		a <u>o</u> b <sup>o</sup> c		a <sup>o</sup> b <sup>o</sup> c		⊐	
Φ	*	a<<b>>c	*	*	*	⊃	<code>liftA2</code>
Φ <sub>1</sub>	*	a<<b>>c	*	*	*	⊃	

Operators

Functions

Trains

Chains

Stacks\*



```
from dovekier 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)))
```

↑      ↖      ↗  
reduce   map



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

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

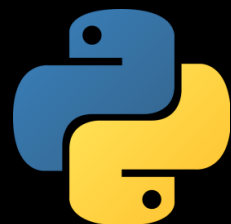


Jello

odd? differ all

## Online REPLS

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



**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](http://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