

DYALOG

Elsinore 2019



Code Golf

– Learn Cutting-Edge APL

Adám Brudzewsky
Marshall Lochbaum

Richard Park · Nicolas Delcros · Richard Smith

DYALOG

Elsinore 2019



Code Golf

– Learn Cutting-Edge APL

Adám Brudzewsky
Marshall Lochbaum

Richard Park · Nicolas Delcros · Richard Smith



Code golf

is a type of recreational computer programming competition in which participants strive to achieve the shortest possible source code...

... known to have been popular with earlier APL hackers.^[Wikipedia]





Code golf

is a type of recreational computer programming competition in which participants strive to achieve the shortest possible source code...

... known to **remain** popular with **today's** APL hackers. [Wikipedia]



Why golf code?

- ◆ Encourages exploration
- ◆ Learn cutting edge APL
- ◆ Be comfortable with lesser known features
- ◆ Achieve excellence
- ◆ Open your mind – think out-of-the-box
- ◆ Have fun!



Goals for Code

- ◆ Readable
- ◆ Fast
- ◆ Short (code golf)
- ◆ Balanced





Goals for Code

- Readable
- Fast
- Short (code golf)
- Balanced

A large red triangle with a thick border, resembling a warning sign, containing the text 'SO, DO NOT GOLF WORK'.

**SO,
DO NOT
GOLF WORK**



Tips for golfing in APL

Check out codegolf.stackexchange.com/q/17665





Tips for golfing in APL

Check out codegolf.stackexchange.com/q/17665



Tips for golfing in APL

Check out codegolf.stackexchange.com/q/17665

\div is your friend:

$$\Leftrightarrow \begin{array}{l} (A < B) \div C \\ C \div \div A < B \end{array}$$


Tips for golfing in APL

Check out codegolf.stackexchange.com/q/17665

\sim is your friend: $\Leftrightarrow \begin{matrix} (A < B) \div C \\ C \div \sim A < B \end{matrix}$

Dealing with $/$ and \neq in trains: $\Leftrightarrow \begin{matrix} \vee / , \\ 1 \in \end{matrix}$



Tips for golfing in APL

Check out codegolf.stackexchange.com/q/17665

\sim is your friend: $\Leftrightarrow \begin{matrix} (A < B) \div C \\ C \div \sim A < B \end{matrix}$

Dealing with $/$ and \neq in trains: $\Leftrightarrow \begin{matrix} \vee / , \\ 1 \in \end{matrix}$

Use \perp : $\Leftrightarrow \begin{matrix} (a \times b) + C \\ a \perp b , C \end{matrix}$



Tips for golfing in APL: *Function Trains*

A 2-train is an *atop*:

$$\begin{array}{lcl} (g\ h)\ \omega & \iff & g\ (h\ \omega) \\ \alpha\ (g\ h)\ \omega & \iff & g\ (\alpha\ h\ \omega) \end{array}$$

A 3-train is a *fork*:

$$\begin{array}{lcl} (f\ g\ h)\ \omega & \iff & (f\ \omega)\ g\ (h\ \omega) \\ \alpha\ (f\ g\ h)\ \omega & \iff & (\alpha\ f\ \omega)\ g\ (\alpha\ h\ \omega) \end{array}$$

The *left tine* of a *fork* (but not an atop!) can be an array:

$$\begin{array}{lcl} (A\ g\ h)\ \omega & \iff & A\ g\ (h\ \omega) \\ \alpha\ (A\ g\ h)\ \omega & \iff & A\ g\ (\alpha\ h\ \omega) \end{array}$$


Ground rules

A solution must be one of:

- ◆ Dfn: $\{\alpha \ \omega\}$ braces *do* count!
- ◆ Derived: $(\ , \circ 0)$ parentheses do *not* count
- ◆ Train: $(\ + \neq \div \neq)$ parentheses do *not* count

Because $f \leftarrow \dots$



SockPairs

Given a list of left (0) and right (1) socks, with N of each in any order, return a shape (N 2) matrix pairing them up.

Example:

L	R		R	R	L	L	L	R
		SockPairs	1	1	0	0	0	1
3	1							
4	2							
5	6							

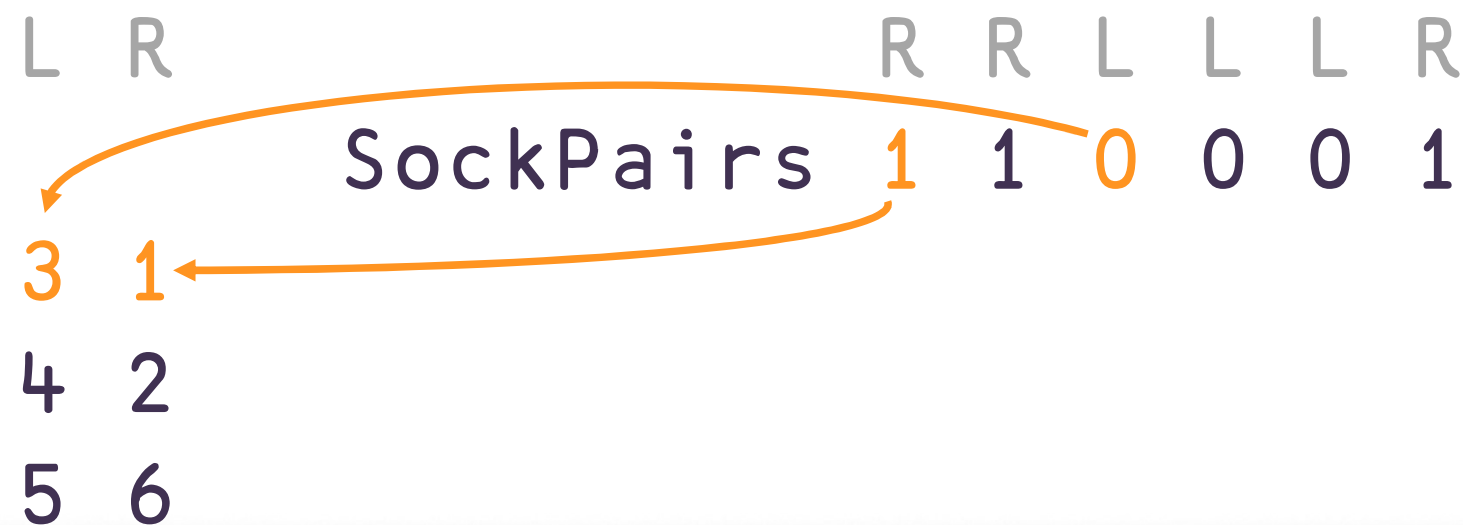
Each row is the index of a left sock followed by the index of a right sock, and each pair should use the left-most socks not already used.



SockPairs

Given a list of left (0) and right (1) socks, with N of each in any order, return a shape (N 2) matrix pairing them up.

Example:



Each row is the index of a left sock followed by the index of a right sock, and each pair should use the left-most socks not already used.



Key operator: *monadic derived function*

$f \overline{\text{vals}}$

For each unique major cell in vals , f is called with the following arguments:

Left: the unique major cell

Right: the indices of all such major cells in vals



Key operator: *monadic derived function*

$\{c\alpha\omega\}$  'Mississippi'

M	1	i	2	5	8	11	s	3	4	6	7	p	9	10
---	---	---	---	---	---	----	---	---	---	---	---	---	---	----



Key operator: *dyadic derived function*

`keys f⌈ vals`

For each unique major cell in `keys`, `f` is called with the following arguments:

Left: the unique major cell from `keys`

Right: the values from `vals` corresponding to all such major cells in `keys`



Key operator: *dyadic derived function*

'Mississippi' {⊢αω}⊞ 10×ι11

M	10	i	20	50	80	110	s	30	40	60	70	p	90	100
---	----	---	----	----	----	-----	---	----	----	----	----	---	----	-----



Key operator: *dyadic derived function*

'Mississippi' {cαω}⊖ 10×ι11

M	10	i	20	50	80	110	s	30	40	60	70	p	90	100
---	----	---	----	----	----	-----	---	----	----	----	----	---	----	-----

'Mississippi' {cαω}⊖ ι≠'Mississippi'

M	1	i	2	5	8	11	s	3	4	6	7	p	9	10
---	---	---	---	---	---	----	---	---	---	---	---	---	---	----



Key operator: *monadic vs dyadic*

$\{c\alpha\omega\} \equiv$ 'Mississippi'

M	1	i	2	5	8	11	s	3	4	6	7	p	9	10
---	---	---	---	---	---	----	---	---	---	---	---	---	---	----

'Mississippi' $\{c\alpha\omega\} \equiv$ $10 \times \iota 11$

M	10	i	20	50	80	110	s	30	40	60	70	p	90	100
---	----	---	----	----	----	-----	---	----	----	----	----	---	----	-----

'Mississippi' $\{c\alpha\omega\} \equiv$ $\iota \neq$ 'Mississippi'

M	1	i	2	5	8	11	s	3	4	6	7	p	9	10
---	---	---	---	---	---	----	---	---	---	---	---	---	---	----



Key operator: *monadic vs dyadic*

$\{c\alpha\omega\} \boxed{\text{'Mississippi'}}$

M	1	i	2	5	8	11	s	3	4	6	7	p	9	10
---	---	---	---	---	---	----	---	---	---	---	---	---	---	----

$f \boxed{\text{keys}} \iff \text{keys } f \boxed{\text{'Mississippi'}}$

$\text{'Mississippi' } \{c\alpha\omega\} \boxed{\text{'Mississippi'}}$

M	1	i	2	5	8	11	s	3	4	6	7	p	9	10
---	---	---	---	---	---	----	---	---	---	---	---	---	---	----



LongestRun

Given a letter and a string, return the length of the longest run in the string which consists entirely of the given letter.

Examples:

```
'x' LongestRun 'xxx,xxxx;xx'
4
'l' LongestRun 'Hello'
2
'y' LongestRun 'Hello'
0
```



LongestRun

Given a letter and a string, return the length of the longest run in the string which consists entirely of the given letter.

Examples:

'x' LongestRun 'xxx,xxxx;xx'
4 ←

'l' LongestRun 'Hello'
2 ←

'y' LongestRun 'Hello'
0 ←



⊆ Partition

Split according to change from previous element in α :

- < increase: new partition
- ≥ decrease/stable: continue partition
- 0= zero: skip element

4 3 0 4 4 4 5 ⊆ 'Hi_APL!'

H i _ A P L !

Hi	APL	!
----	-----	---



⊆ Partition

Split according to change from previous element in α :

< increase: new partition

≥ decrease/stable: continue partition

0= zero: skip element

0 <4 ≥3 0=0 <4 ≥4 ≥4 <5 ⊆ 'Hi_APL!'

H i _ A P L !

Hi	APL	!
----	-----	---



⊆ Partition

Split according to change from previous element in α :

< increase: new partition

≥ decrease/stable: continue partition

0= zero: skip element

0 <4 ≥3 0=0 <4 ≥4 ≥4 <5 ⊆ 'Hi_APL!'

H i _ A P L !

Hi

APL

!



⊆ Partition

Split according to change from previous element in α :

< increase: new partition

≥ decrease/stable: continue partition

0= zero: skip element

0 < 4 ≥ 3 0=0 < 4 ≥ 4 ≥ 4 < 5 ⊆ 'Hi_APL!'

H i _ A P L !

Hi	APL	!
----	-----	---



⊆ Partition

Split according to change from previous element in α :

< increase: new partition

≥ decrease/stable: continue partition

0= zero: skip element

0 < 4 ≥ 3 0=0 < 4 ≥ 4 ≥ 4 < 5 ⊆ 'Hi_APL!'

H i _ A P L !

Hi APL !



⊆ Partition

Split according to change from previous element in α :

< increase: new partition

≥ decrease/stable: continue partition

0= zero: skip element

0 < 4 ≥ 3 0=0 < 4 ≥ 4 ≥ 4 < 5 ⊆ 'Hi_APL!'

H i _ A P L !

Hi	APL	!
----	-----	---



⊆ Partition: *Boolean* α

Split according to change from previous element in α :

< increase: new partition

= decrease/stable: continue partition

0= zero: skip element

1	1	0	1	1	1	1	⊆	'Hi_APL!'
H	i	_	A	P	L	!		

Hi	APL!
----	------



⊆ Partition: *Boolean* α

Split according to change from previous element in α :

< increase: new partition

= decrease/stable: continue partition

0= zero: skip element

0 < 1 ≥ 1 0=0 < 1 ≥ 1 ≥ 1 ≥ 1 ⊆ 'Hi_APL!'

H i _ A P L !

Hi	APL!
----	------



⊆ Partition: *Boolean* α

Split according to change from previous element in α :

< increase: new partition

= decrease/stable: continue partition

0= zero: skip element

0 < 1 ≥ 1 0=0 < 1 ≥ 1 ≥ 1 ≥ 1 ⊆ 'Hi_APL!'

H i _ A P L !

Hi

APL!



⊆ Partition: *Boolean* α

Split according to change from previous element in α :

< increase: new partition

= decrease/stable: continue partition

0= zero: skip element

0 < 1 ≥ 1 0=0 < 1 ≥ 1 ≥ 1 ≥ 1 ⊆ 'Hi_APL!'

H i _ A P L !

Hi	APL!
----	------



⊆ Partition: *Boolean* α

Split according to change from previous element in α :

< increase: new partition

= decrease/stable: continue partition

0= zero: skip element

0 < 1 ≥ 1 0=0 < 1 ≥ 1 ≥ 1 ≥ 1 ⊆ 'Hi_APL!'

Hi _ A P L !

Hi APL!



⊆ Partition: *Boolean* α

Split according to change from previous element in α :

< increase: new partition

= decrease/stable: continue partition

0= zero: skip element

0 < 1 ≥ 1 0=0 < 1 ≥ 1 ≥ 1 ≥ 1 ⊆ 'Hi_APL!'

H i _ A P L !

Hi	APL!
----	------



⊆ Partition: *Boolean* α

Split according to change from previous element in α :

< increase: new partition

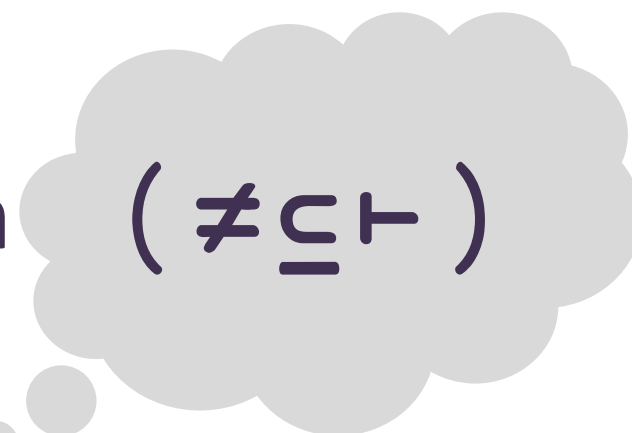
= decrease/stable: continue partition

0= zero: skip element

0 < 1 ≥ 1 0=0 < 1 ≥ 1 ≥ 1 ≥ 1 ⊆ ··· 'Hi_APL!'

H i _ A P L !

Hi	APL!
----	------



⊆ Partition: *Boolean* α

Split value:

1= one: new/continue partition

0= zero: skip element

1=**1** **1**=**1** **0**=**0** **1**=**1** **1**=**1** **1**=**1** **1**=**1**
H **i** **_** **A** **P** **L** **!**

Hi	APL!
----	------

($\neq \subseteq \vdash$)
 ⊆ · 'Hi_APL!'



□DL□



DateStamp

Given a timestamp in \square TS form
(Y M D h m s t), return a 10-element
character vector like 'YYYY-MM-DD'.
You may assume that the year has four digits.

Examples:

```
DateStamp 1966 11 27 15 53 59 0  
1966-11-27
```

```
DateStamp  $\square$ TS  
2019-09-12
```



DateStamp

Given a timestamp in `⌈TS` form
(Y M D h m s t), return a 10-element
character vector like 'YYYY-MM-DD'..
You may assume that the year has four digits.

Examples:

```
DateStamp 1966 11 27 15 53 59 0
```

```
1966-11-27
```

```
DateStamp ⌈TS
```

```
2019-09-12
```

Richard Smith's
'YYYY-MM-DD' ⋅ ⌈DT
in 18.0



DateStamp

Given a timestamp in `⌈TS` form
(Y M D h m s t), return a 10-element
character vector like 'YYYY-MM-DD'...
You may assume that the year has four digits.

Examples:

```
DateStamp 1966 11 27 15 53 59 0
1966-11-27
```

```
DateStamp ⌈TS
2019-09-12
```

Richard Smith's
'YYYY-MM-DD' ⋅ ⌈DT
in 18.0



@ At operator

```
(mod @ sel) ω  ⍝ say: "mod at sel in ω"
```

Right operand **sel** selects items of ω to modify (or replace).

Left operand **mod** says how to modify them (or what to replace them with).

```
100 2 300 4 500 6 700      ⍝ "100-fold at odd items"
      100 2 300 4 500 6 700
      ('*' @ 2 4) 'Hello'  ⍝ "* at 2 4" (replacement)
H* l* o
```



@ At operator: *operands*

Left operand (modifier) can be either:

- An *array* of replacement values
- A *function* used to modify values

Right operand (selector) can be either:

- A *simple array* selecting major cells
- A *nested array* for choose or reach indexing
- A *function*, applied to ω , returning a Boolean mask used for scatter-point indexing



@ At operator: *left operand (modifier)*

┌────────── Left operand ─────────┐

array of replacement values

function to modify values

 (200 400@2 4) ⍲5 A array

1 200 3 400 5

 (ϕ@2 4) ⍲5 A monadic function

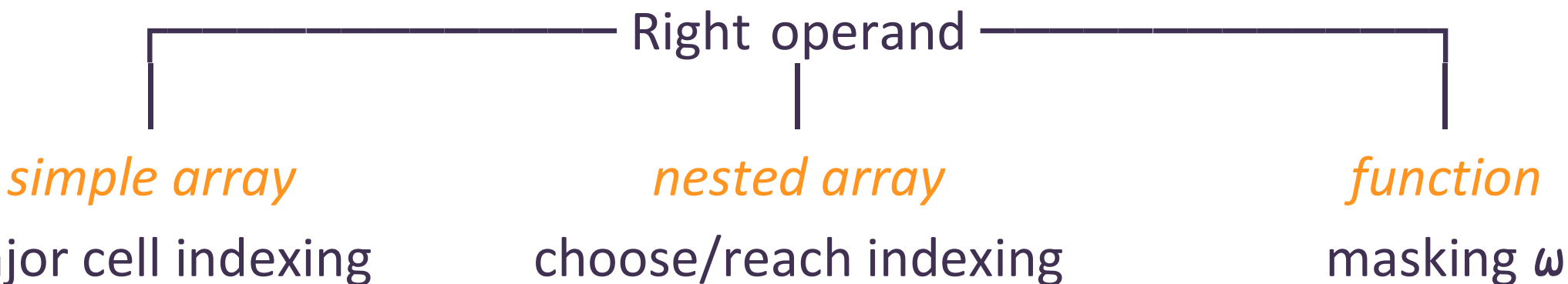
1 4 3 2 5

 100 (×@2 4) ⍲5 A dyadic function

1 200 3 400 5



@ At operator: *right operand (selector)*



$\neg 10 @ 1 \ 3 \ 5 \ 19$ a simple array
 $\neg 10 \ 2 \ \neg 10 \ 4 \ \neg 10 \ 6 \ 7 \ 8 \ 9$
 $\neg 10 @ (2 \circ |) \ 2 \ 3 \rho 16$ a Boolean function
 $\neg 10 \quad 2 \ \neg 10$
 $4 \ \neg 10 \quad 6$



@ At operator: *a potential parsing problem*

Array right operand: `⊢⋆9 'Hello'`



@ At operator: *a potential parsing problem*

Array right operand: $\leftarrow \times 9$ 'Hello'

parses as: $\leftarrow \times (9$ 'Hello')



@ At operator: *a potential parsing problem*

Array right operand: $\leftarrow \ddot{*} 9 \text{ 'Hello'}$

parses as: $\leftarrow \ddot{*} (9 \text{ 'Hello'})$

So, use parentheses: $(\leftarrow \ddot{*} 9) \text{ 'Hello'}$

Or end of statement: $f \leftarrow \ddot{*} 9 \diamond f \text{ 'Hello'}$



@ At operator: *a potential parsing problem*

Array right operand: $\leftarrow \times 9 \text{ 'Hello'}$

parses as: $\leftarrow \times (9 \text{ 'Hello'})$

So, use parentheses: $(\leftarrow \times 9) \text{ 'Hello'}$

Or end of statement: $f \leftarrow \leftarrow \times 9 \diamond f \text{ 'Hello'}$

Golf with identity fn: $\leftarrow \times 9 \vdash \text{ 'Hello'}$

Or another function: $\leftarrow \times 9 \phi \text{ 'Hello'}$



Stencil Operator

(f  settings) data

2

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

(f  settings) data

2

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

(f  settings) data

2

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

(f  settings) data

2

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

(f  settings) data

2 2

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

(f  settings) data

2 2

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

( settings) data

2 2
1 1

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

( settings) data

2 2
1 1

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

(  settings) data

2 2
1 1

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

(  settings) data

2 2
2 2

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

( settings) data

2 2
2 2

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

(  settings) data

2 2
2 2

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

(f  settings) data

2 2
2 2

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3



Stencil Operator

1 1 (f settings) data

3 3
1 1

0	0	0	0	0	0
0	3	1	4	1	0
0	5	9	2	6	0
0	5	3	5	8	0
0	9	7	9	3	0
0	0	0	0	0	0



⌈ Stencil Operator

1 0 (f⌈settings) data

3 3
1 1

0	0	0	0	0	0
0	3	1	4	1	0
0	5	9	2	6	0
0	5	3	5	8	0
0	9	7	9	3	0
0	0	0	0	0	0



⌈ Stencil Operator

1 0 (f ⌈ settings) data

3 3

0	0	0	0	0	0
0	3	1	4	1	0
0	5	9	2	6	0
0	5	3	5	8	0
0	9	7	9	3	0
0	0	0	0	0	0



Stencil Operator

1 -1 (f settings) data

3 3

0	0	0	0	0	0
0	3	1	4	1	0
0	5	9	2	6	0
0	5	3	5	8	0
0	9	7	9	3	0
0	0	0	0	0	0



⌈ Stencil Operator

0 1 (f ⌈ settings) data

3 3

0	0	0	0	0	0
0	3	1	4	1	0
0	5	9	2	6	0
0	5	3	5	8	0
0	9	7	9	3	0
0	0	0	0	0	0



⌈ Stencil Operator

0 0 (f ⌈ settings) data

3 3

0	0	0	0	0	0
0	3	1	4	1	0
0	5	9	2	6	0
0	5	3	5	8	0
0	9	7	9	3	0
0	0	0	0	0	0



DiseaseSpread

Given a Boolean matrix world, generate the next iteration where:

- any cell which shared a *side* (not a *corner*) with an infected cell becomes infected.
- the infected cells stay infected forever.

Example:

←world←2⊥*~1←6↑⊠AVU

0	0	0	0	1	0
0	0	0	0	0	0
0	1	1	1	0	1
0	0	0	1	0	1
0	0	1	0	0	0
0	0	0	1	0	0

DiseaseSpread world

0	0	0	1	1	1
0	1	1	1	1	1
1	1	1	1	1	1
0	1	1	1	1	1
0	1	1	1	0	1
0	0	1	1	1	0



DiseaseSpread

Given a Boolean matrix world, generate the next iteration where:

- any cell which shared a *side* (not a *corner*) with an infected cell becomes infected.
- the infected cells stay infected forever.

Example:

←world←2⊥*~1←6↑⊠AVU

0	0	0	0	1	0
0	0	0	0	0	0
0	1	1	1	0	1
0	0	0	1	0	1
0	0	1	0	0	0
0	0	0	1	0	0



DiseaseSpread world

0	0	0	1	1	1
0	1	1	1	1	1
1	1	1	1	1	1
0	1	1	1	1	1
0	1	1	1	0	1
0	0	1	1	1	0



Timeline: *Golfing Features*

14.0 $f\boxed{\text{key}}$ Key $f\ddot{o}k$ Rank $(+/\div 1\lceil\neq)$ function trains

15.0 File functions: $\boxed{N}INFO$ $\boxed{N}GET$ $\boxed{N}PUT$ etc.

16.0 $\subseteq Y$ Nest $X\subseteq Y$ Partition $\underline{1}Y$ Where $X\underline{1}Y$ Interval Index
 $f@...$ At $f\boxed{\text{key}}$ Stencil $\boxed{N}CSV$ $\boxed{N}JSON$

17.0 TAO: ΔY and ΨY $X\underline{1}Y$ $\cup Y$ on high rank

18.0 $f\ddot{o}g$ Atop $f\ddot{o}g$ Over $A\ddot{\sim}$ Constant



Code Golf Stack Exchange

- codegolf.stackexchange.com (you can search for **apl tips**)
- Use [tio.run](#) to automatically format a submission
- Explain your code if you can
- You may use `⎕` and `⎕` for input ("Full program"=tradfn body)



Code Golf Stack Exchange

- codegolf.stackexchange.com (you can search for **apl tips**)
- Use [tio.run](#) to automatically format a submission
- Explain your code if you can
- You may use `⎕` and `⎕` for input ("Full program"=tradfn body)

`DateStamp←' - '@5 8∘⌥1E3⊥3↑⌵`



Code Golf Stack Exchange

- codegolf.stackexchange.com (you can search for **apl tips**)
- Use [tio.run](#) to automatically format a submission
- Explain your code if you can
- You may use `⎕` and `⎕` for input ("Full program"=tradfn body)

`DateStamp` ← ' - ' @5 8 ° ¤ 1 E 3 ⊥ 3 ↑ ⋈

▽ `DateStamp`

' - ' @5 8 ¤ 1 E 3 ⊥ 3 ↑ ⎕

▽

