ZigZag

A CHAUTAUQUA ON J

DR. JOSHUA ECKROTH
STETSON UNIVERSITY

Syntax

- Statements are exactly one line long
- Numbers may be written 2, _2, 0.5, 5, 1e3, 2j5, 2r7, 16b1f, 2p1, _, __
- Adjacent numbers make a list: 3 5 2e3 7
- Strings have single-quotes: 'hello'
- Everything else is a built-in or user-defined word:
  - Verbs: +, -, %, etc.
  - Adverbs: ~, /, }, etc.
  - Conjunctions: ^:, ", @, &, etc.
  - Others: =., =:, NB., etc.

You do not need to trouble yourself with the distinction between integers, floats, and complex numbers. If it's a number, J will handle it properly.

J for C Programmers

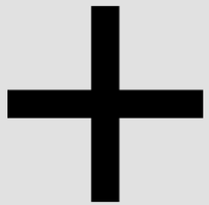
Evaluation Rules

Forget the table of operator precedence!

J for C Programmers

RTL Evaluation:

```
      2*4+5
18
      2+4*5
22
      (2*4)+5
13
```



+ 0 0 0

Conjugate

+ y is the conjugate of y . For example, +3j4 is 3j_4 .

Plus

+ is defined as in elementary arithmetic, and is extended to complex numbers as usual.

Verb

- 0 0 0

Negate

-y is the negative of y . That is, it is defined as 0 - y . Thus, -2 0 _2 is _2 0 2 .

Minus

- is defined as in elementary arithmetic, and is extended to complex numbers as usual.

Verb

* 0 0 0

Signum

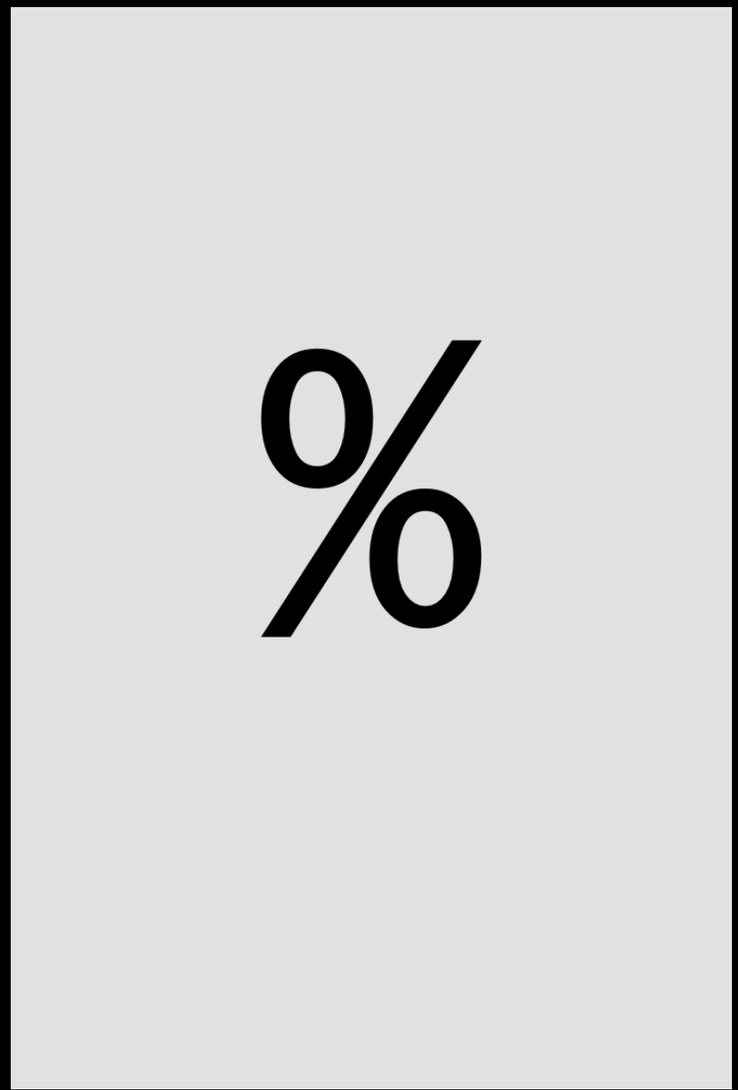
*y is _1 if y is negative, 0 if it is zero, 1 if it is positive; more generally, *y is the intersection of the unit circle with the line from the origin through the argument y in the complex plane.

Times

* denotes multiplication, defined as in elementary mathematics and extended to complex numbers as usual.

Verb

%



% 0 0 0

Reciprocal

`% y` is the reciprocal of `y` , that is, `1%y` .

Divided by

`x % y` is division of `x` by `y` as defined in elementary math, except that `0%0` is `0` .

Verb

Monads & Dyads

%

% 0 0 0

Reciprocal

```
% y is the
reciprocal of
y , that is,
1%y .
```

y is right arg

Divided by

```
x % y is
division of x
by y as
defined in
elementary
math, except
that 0%0 is
0 .
```

x is left arg,
y is right arg

Verb

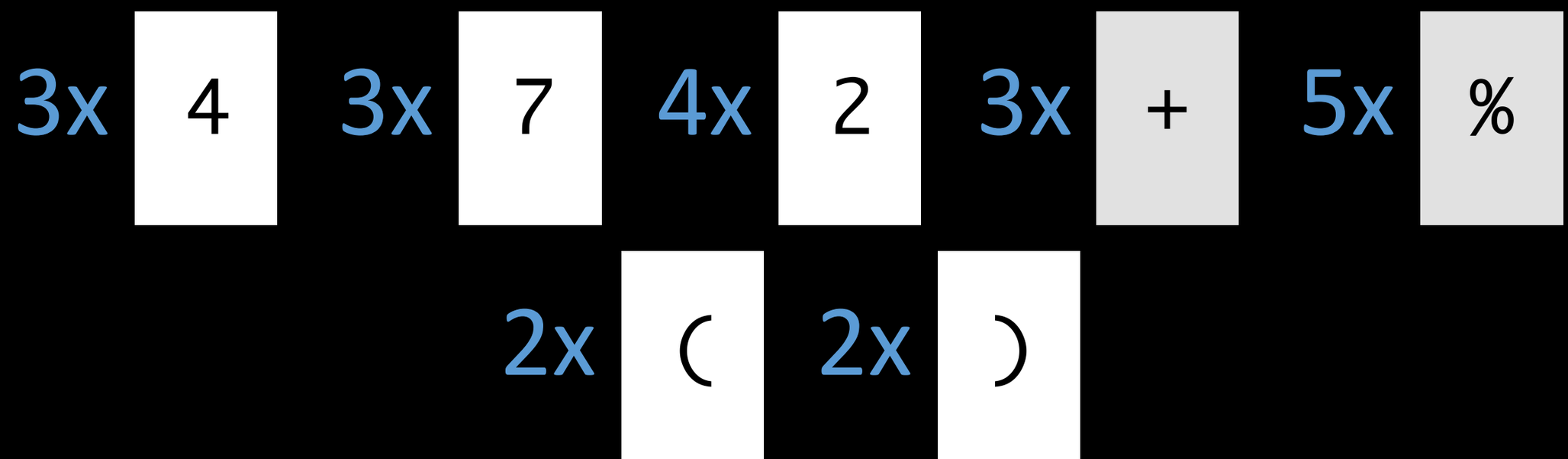
Monad vs. Dyad %:

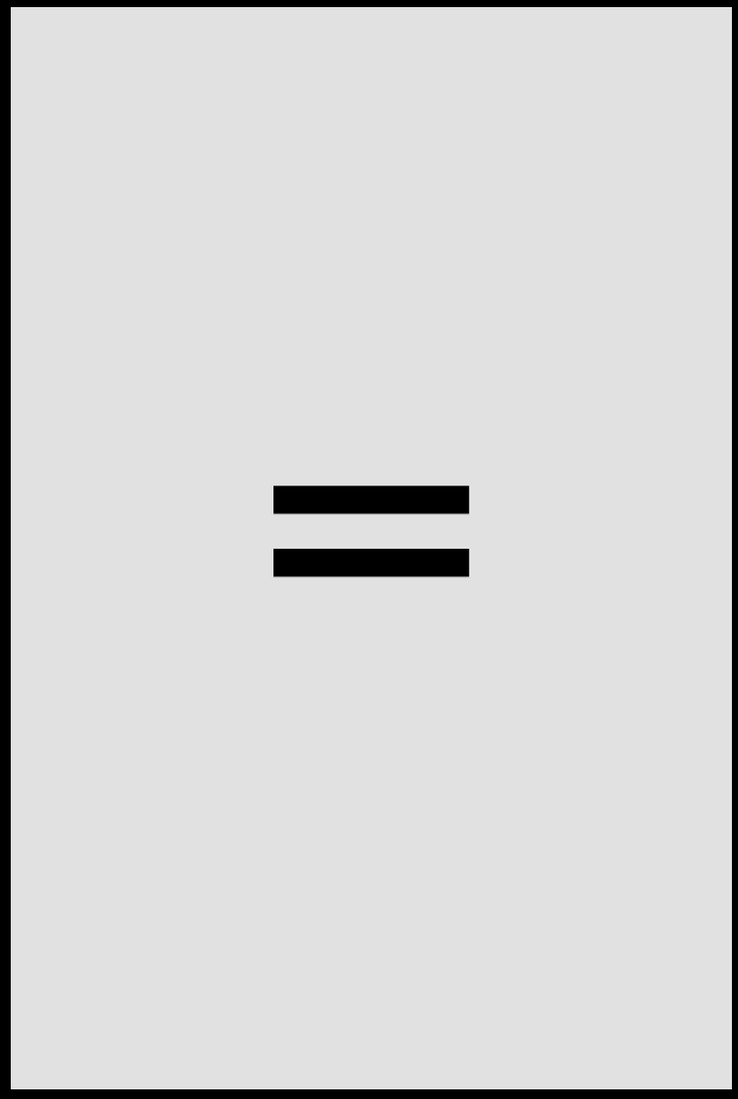
```
    2%7
0.285714
    7%2
3.5
    %7
0.142857
    %2
0.5
```

Challenge!

Compute the arithmetic mean of 4 and 7.

Find <u>three</u> solutions using all of these cards simultaneously:





= _ 0 0

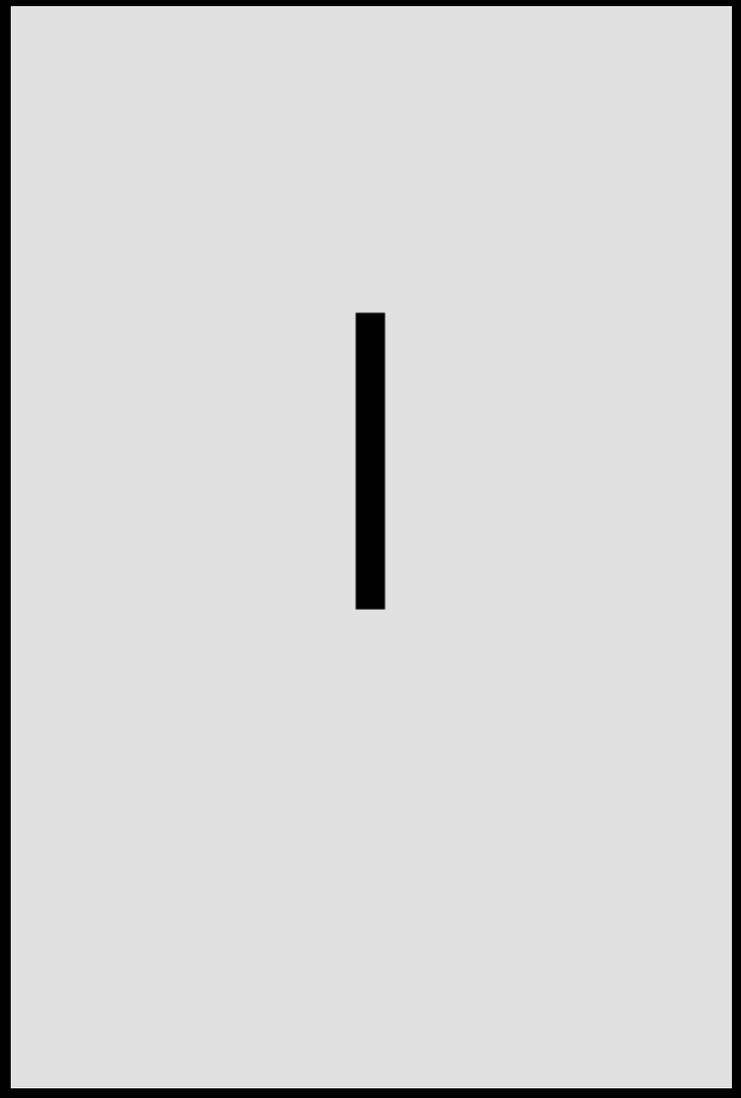
Self-Classify

=y classifies the items of the nub of y (that is, ~.y) according to equality with the items of y , producing a boolean table of shape #~.y by #y .

Equal

x=y is 1 if x is equal to y , and is otherwise 0 .

Verb



| 0 0 0

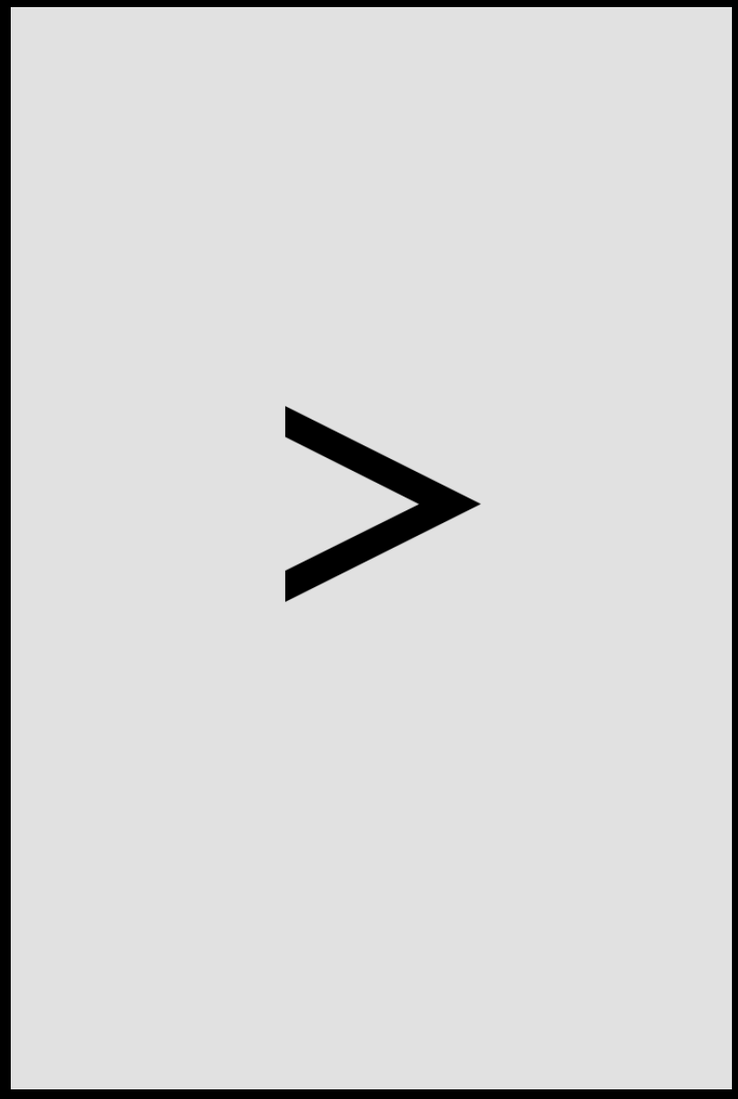
Magnitude

```
|y == %:y*+y .
```

Residue

The familiar use of residue is in determining the remainder on dividing a non-negative integer by a positive.

Verb



> 0 0 0

Open

Open is the inverse of box, that is, ><y is y . When applied to an open array (that has no boxed elements), open has no effect. Opened atoms are brought to a common shape.

Larger Than

x>y is 1 if x is tolerantly larger than y . Tolerance t is provided by >!.t .

Verb

```
~: _ 0 0
```

Nub Sieve

```
~:y is the
boolean list b
such that b#y
is the nub of
y .
```

Not Equal

```
x~:y is 1 if x
is tolerantly
unequal to y .
```

Verb

-. 0 _ _

Not

-.y is 1-y ; for a boolean argument it is the complement (not); for a probability, it is the complementary probability.

Less

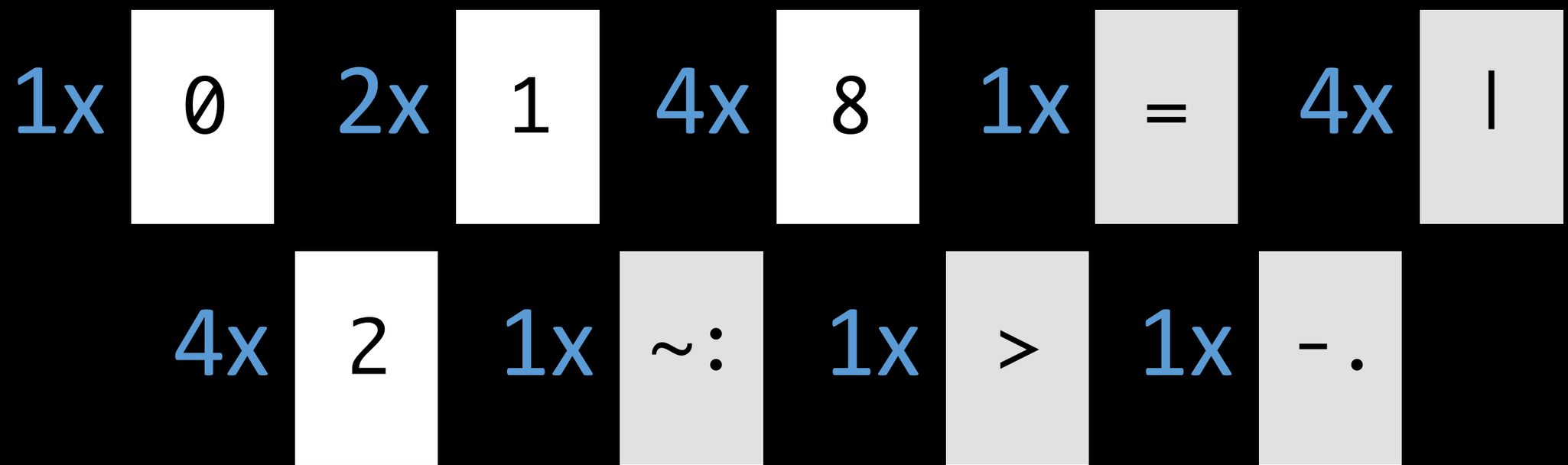
x-.y includes all items of x except for those that are cells of y .

Verb

Challenge!

Compute whether a number, say 8, is even (gives 1 if even, 0 otherwise).

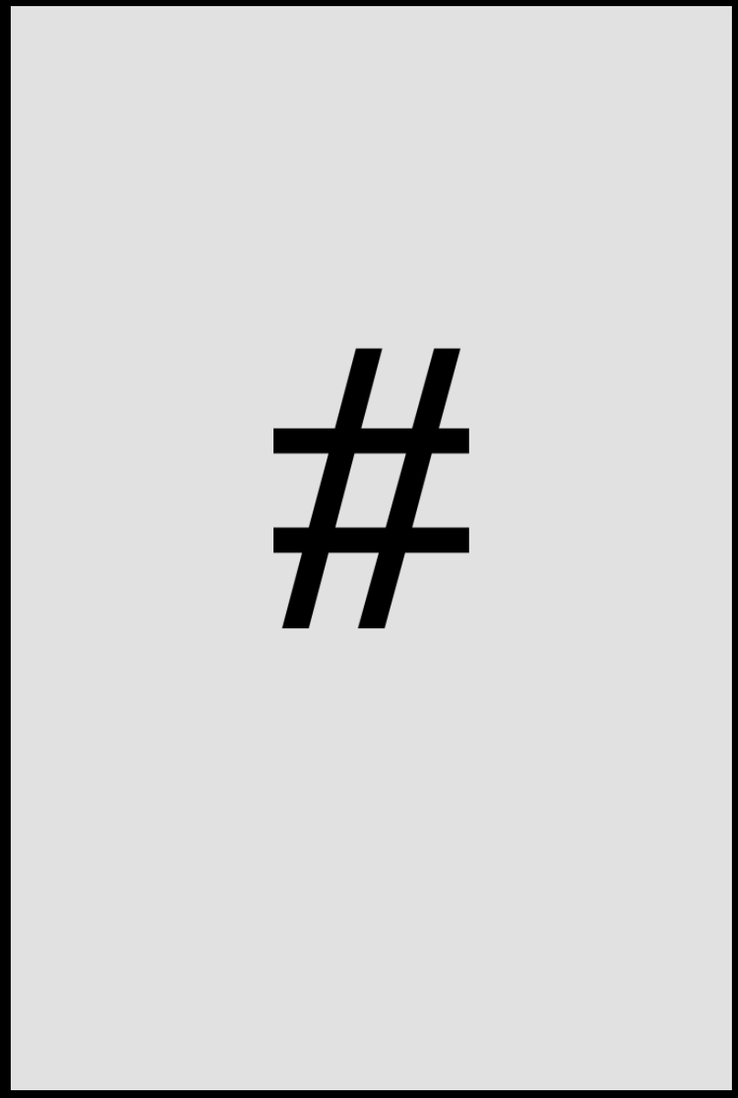
Find four solutions using all of these cards simultaneously:



Arrays

Any variable can be an array. As for what the type and dimensioning is: you assigned the variable, didn't you? It contains whatever you put into it, a number, a string, an array, a structure... J will remember. If your program logic requires you to find out the current attributes of a variable, J can tell you.

J for C Programmers



_ 1 _

Tally

#y is the number of items in y .

Copy

If the arguments have an equal number of items, then x#y copies +/x items from y, with i{x repetitions of item i{y . Otherwise, if one is an atom it is repeated to make the item count of the arguments equal.

Verb

Variable assignment and array size:

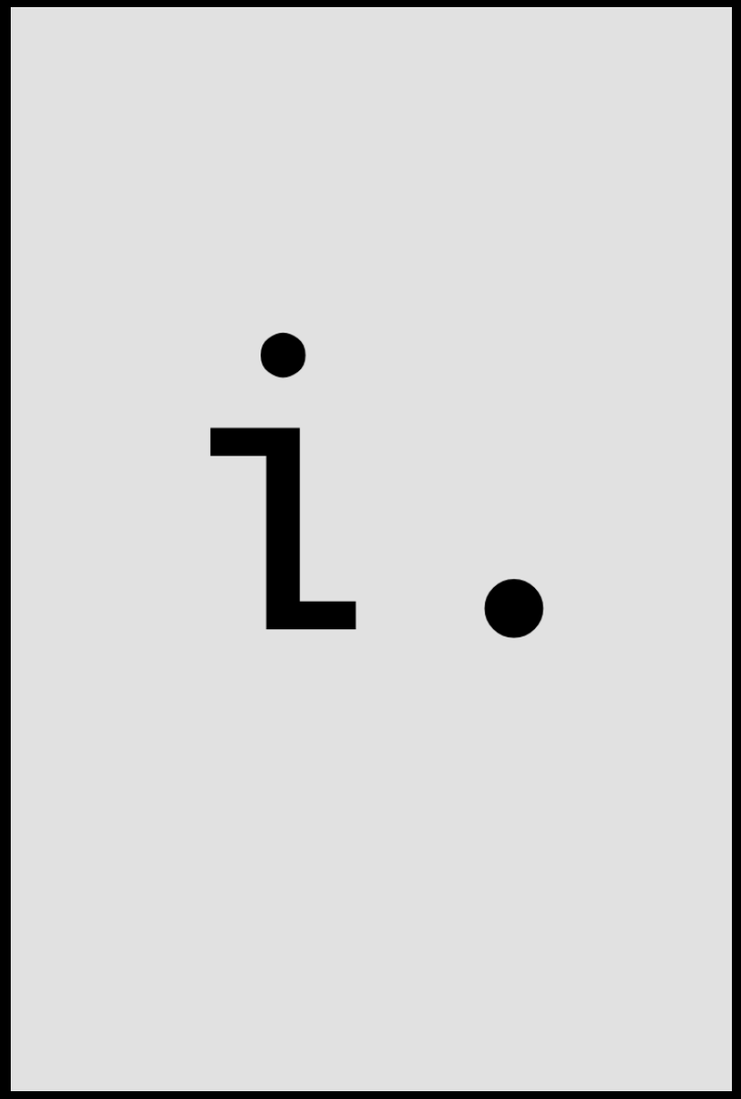
```
      a =: 5 3 7 9
      a
5 3 7 9
      #a
4
      s =: 'hello'
      #s
5
```

In J, every operator has a loop built in.

J for C Programmers

Automatic element-by-element operations:

```
    1 2 3 + 4 5 6
5 7 9
    1 + 4 5 6
5 6 7
    1 2 + 4 5 6
|length error
|   1 2    +4 5 6
```



i. 1 _ _

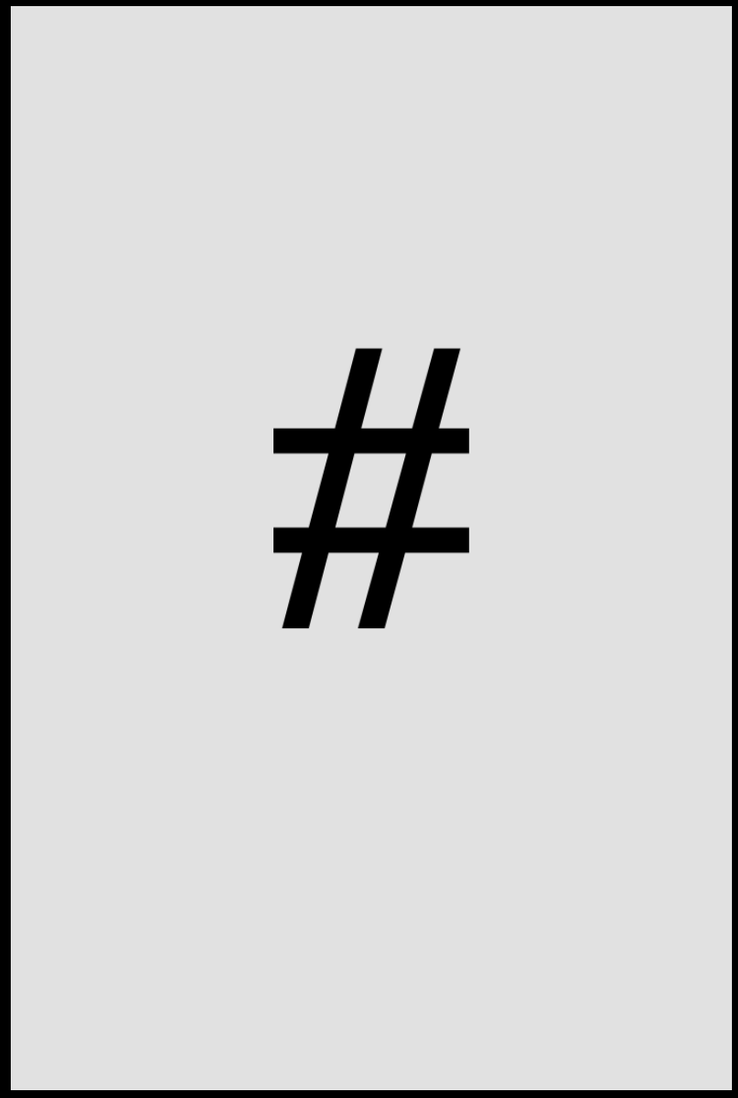
Integers

The shape of i.y is |y , and its atoms are the first */|y non-negative integers. A negative element in y causes reversal of the atoms along the corresponding axis.

Index Of

If rix is the rank of an item of x, then the shape of the result of x i. y is (-rix)}. $y . Each atom of the result is either #x or the index of the first occurrence among the items of x of the corresponding rix-cell of y .

Verb



_ 1 _

Tally

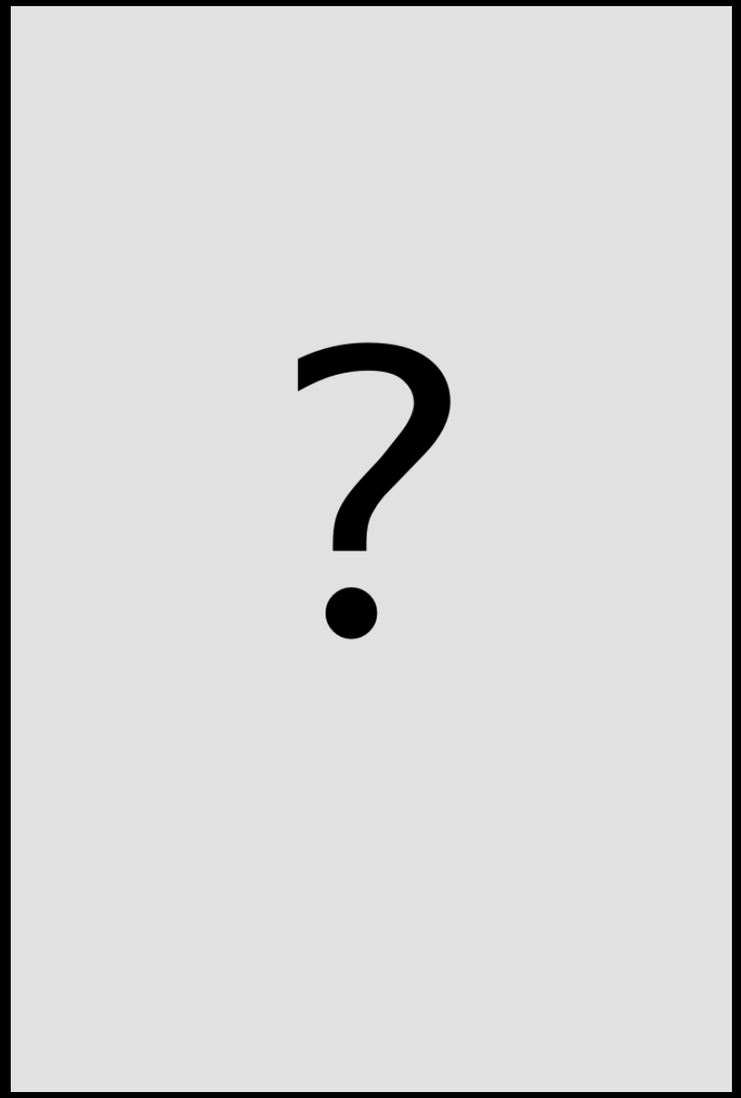
#y is the number of items in y .

Copy

If the arguments have an equal number of items, then x#y copies +/x items from y, with i{x repetitions of item i{y . Otherwise, if one is an atom it is repeated to make the item count of the arguments equal.

Verb

```
     i.4
0 1 2 3
     3+i.4
3 4 5 6
     3#i.4
0 0 0 1 1 1 2 2 2 3 3 3
     #3#i.4
12
     3##i.4
4 4 4
```



? 0 0 0

Roll

`? y` yields a uniform random selection from the population `i.y` if `y` is a positive integer, or from the interval of numbers greater than `0` and less than `1` if `y` is `0`.

Deal

`x ? y` is a list of `x` items randomly chosen without repetition from `i.y` .

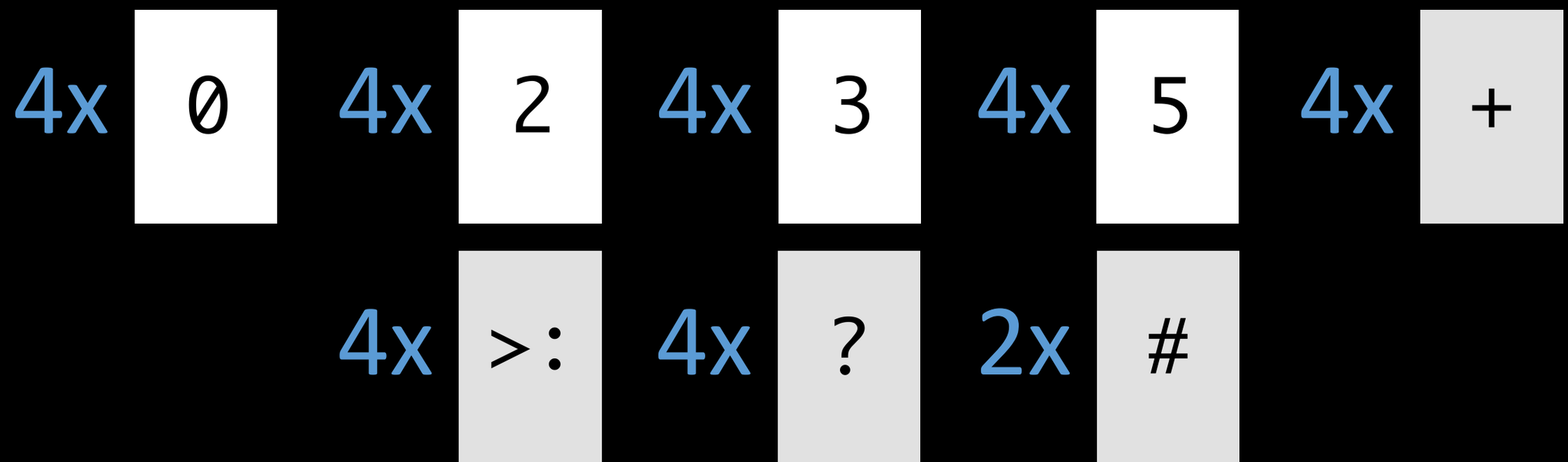
Verb

```
    ?5
1
    ?5
4
    ?0
0.654682
    ?5 2 3 6
3 1 0 4
    3?7
1 4 2
```

Challenge!

Roll five d20 dice with +3 modifiers on all rolls.

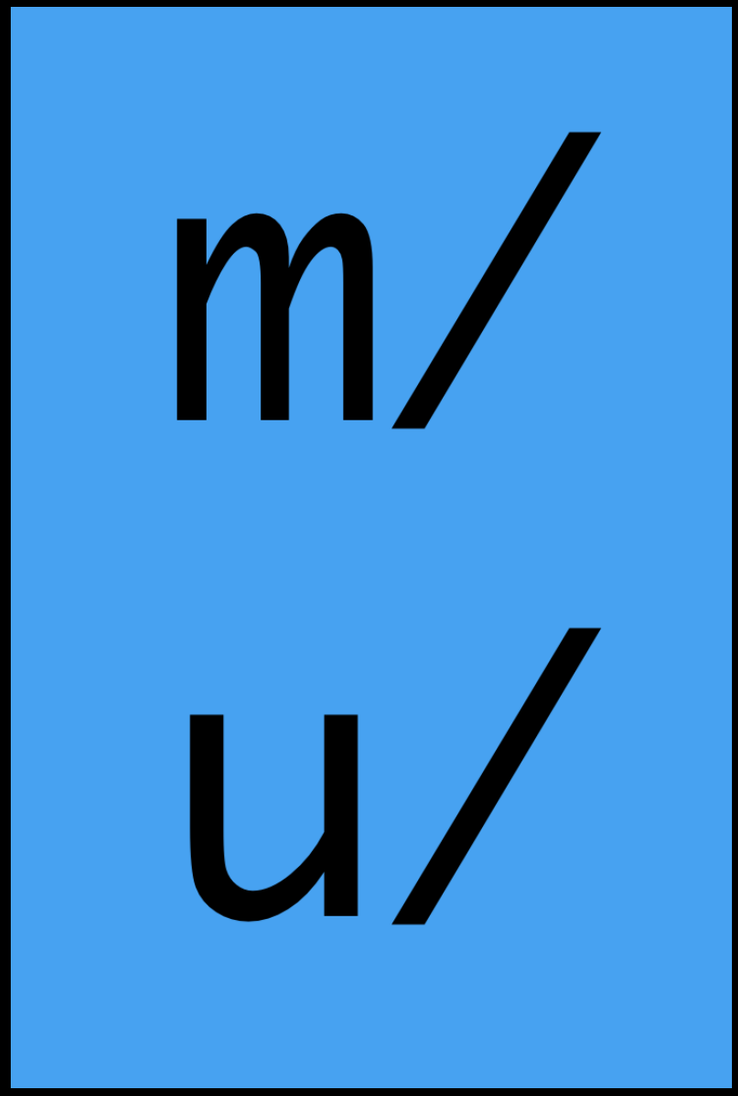
Find four solutions using all of these cards simultaneously:



Adverbs

A noun holds data; a verb operates on one or two nouns to produce a result which is a noun; an adverb operates on one noun or verb to produce a derived entity.

J for C Programmers



m/ u/ _ _ _

Insert

u/y applies the dyad u between the items of y . m/y inserts successive verbs from the gerund m between items of y, extending m cyclically as required. Thus, +`*/i.6 is 0+1*2+3*4+5 .

Table

If x and y are numeric lists, then x */ y is their multiplication table.

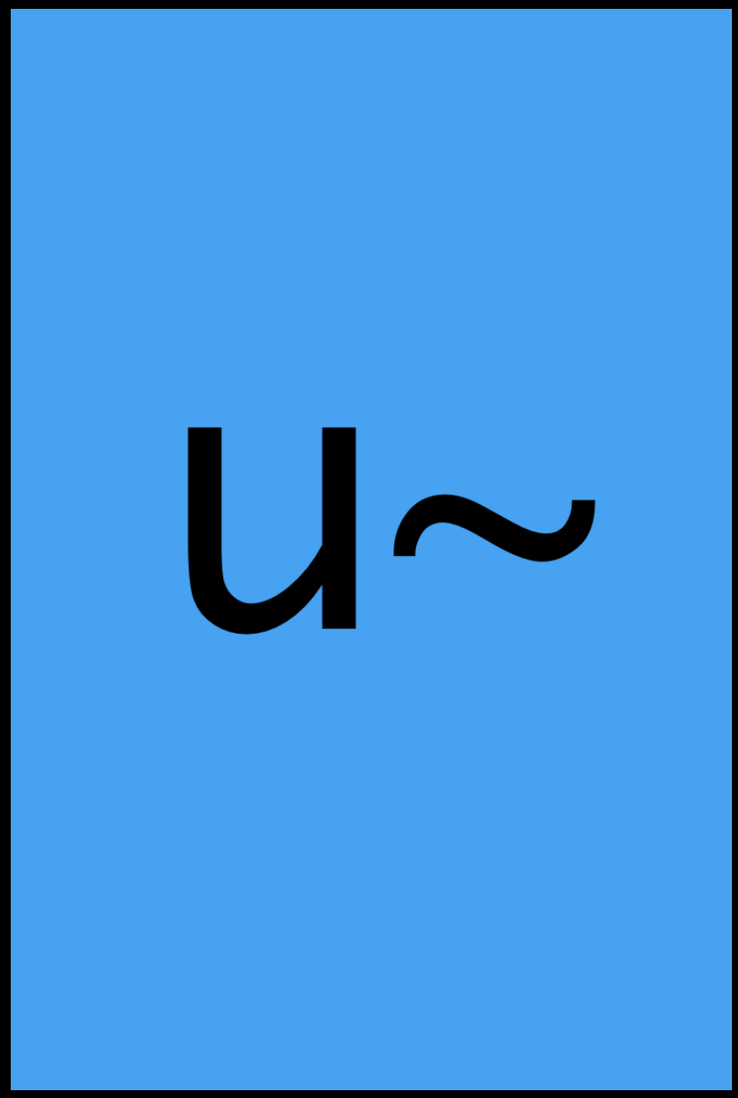
Adverb

```
    1+2+3+4+5
15
    +/1 2 3 4 5
15
    */1 2 3 4 5
120
```

New version of “arithmetic mean”:

```
    vals=:3 _4 7 9 _2 3
    +/vals
16
    #vals
6
    (+/vals)%(#vals)
2.66667
    +/vals%#vals
2.66667
```

u~



u~ _ ru lu

Reflexive

u~ y == y u y .
For example, ^~ 3 is 27, and +/ ~ i. n is an addition table.

Passive

~ commutes or crosses connections to arguments: x u~ y == y u x .

Adverb

{ 1 0 _

Catalogue

{y forms a catalogue from the atoms of its argument, its shape being the chain of the shapes of the opened items of y . The common shape of the boxed results is $y .

From

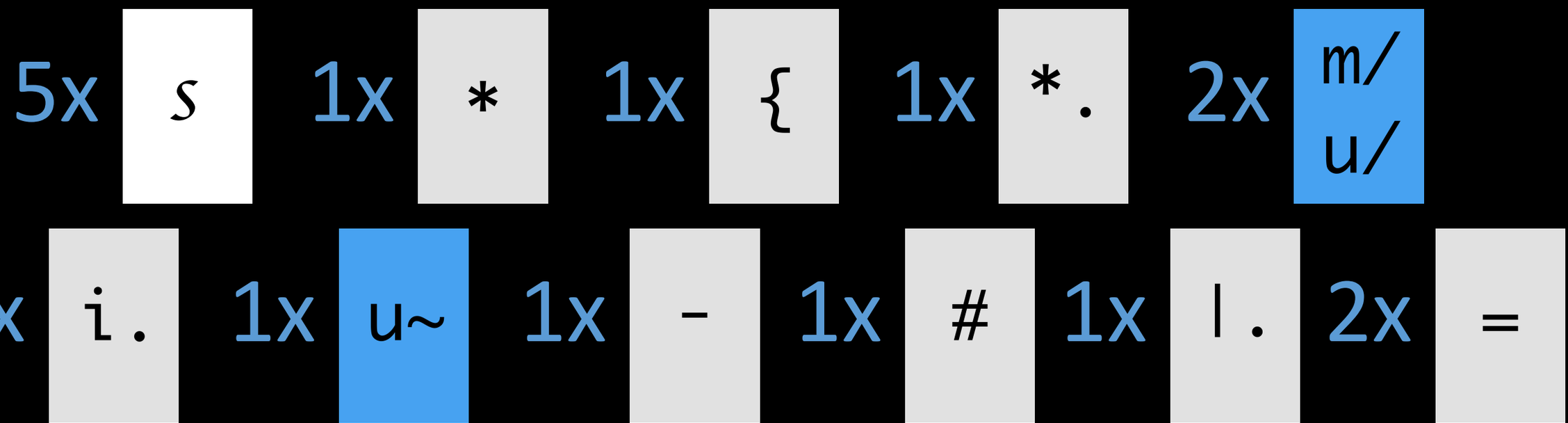
If x is an integer in the range from -n=: #y to n-1 , then x{y selects item n|x from y .

Verb

Challenge!

Given a string (or any array) s, determine if it's a palindrome.

Find <u>two</u> solutions using all of these cards simultaneously:



Hooks & Forks

Hooks `fg`

Monadic

```
fg y == y f (g y)
-% y == y - (% y)
```

Dyadic

```
x fg y == x f (g y)
x -% y == x - (% y)
```

```
      f =: monad : '0=2|y'    NB. checks that y is even
      i.10
0 1 2 3 4 5 6 7 8 9
      f i.10
1 0 1 0 1 0 1 0 1 0

      NB. use # operator to keep only evens
      (f i.10) # i.10
0 2 4 6 8
      NB. use a hook
      (#~f) i.10          NB. i.e., i.10 #~ (f i.10)
0 2 4 6 8                 NB. i.e., (f i.10) # i.10
```

Forks `fgh`

Monadic

```
fgh y == (f y) g (h y)
-+% y == (- y) + (% y)
```

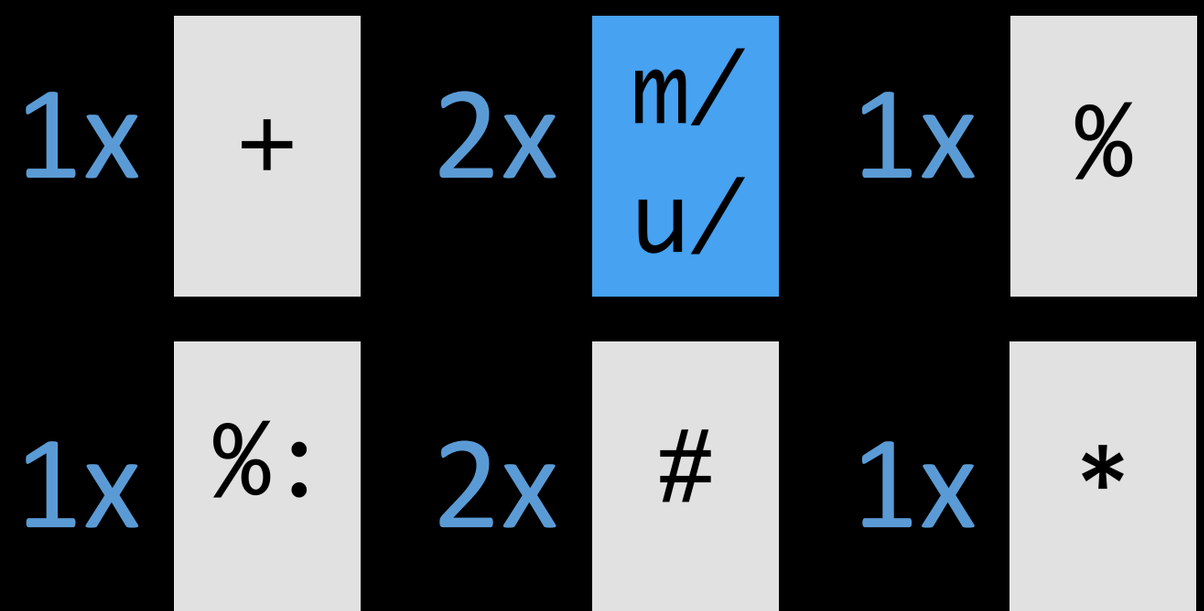
Dyadic

```
x fgh y == (x f y) g (x h y)
x -+% y == (x - y) + (x % y)
```

Challenge!

Define the arithmetic mean and geometric mean for any list of numbers. Geometric mean = $(x_1*x_2*\ldots*x_n)^{(1/n)}$.

Find one solution for each kind of mean using all of these cards simultaneously:



Conjunctions

J uses the names x, y, u, v, m, and n to represent arguments to verbs and other entities. You should avoid using these names for other purposes.

J for C Programmers

m&v

u&n

m&v u&n _ 0 _

Bond

m&v y is defined as m v y ; that is, the left argument m is bonded with the dyad v to produce a monadic function. Similarly, u&n y is defined as y u n ; in other words, as the dyad u provided with the right argument n to produce a monadic function.

```
x m&v y ==
m&v^:x y
x u&n y ==
u&n^:x y
```

Conjunction

```
     f =: 5&+
     f 1 2 3 4 5
6 7 8 9 10
     g =: -&5
     g 1 2 3 4 5
_4 _3 _2 _1 0
```

u&v

u&v mv mv mv

Compose

```
u&v y == u v y . Thus +:&- 7 is _14 (double the negation). Moreover, the monads u&v and u@v are equivalent.
```

```
x u&v y == (v x) u (v y) . For example, 3 +&! 4 is 30, the sum of factorials.
```

Conjunction

```
     f =: -&%                          NB. i.e., -(%y)
     f 1 2 3 4 5
_1 _0.5 _0.333333 _0.25 _0.2
     10 f 1 2 3 4 5                    NB. i.e., (%x)-(%y)
_0.9 _0.4 _0.233333 _0.15 _0.1

     g =: -%                           NB. compare with hook
     g 1 2 3 4 5                       NB. i.e., y-(%y)
0 1.5 2.66667 3.75 4.8
     10 g 1 2 3 4 5                    NB. i.e., x-(%y)
9 9.5 9.66667 9.75 9.8
```

u@:v

u@:v _ _ _

At

```
u@:v y == u v y .
For example,
+:@:- 7 is _14
(double the
negation).
Moreover, the
monadic uses of
u@v and u&v are
equivalent.
```

```
x u@:v y == u x v
y . For example,
3 +:@:- 7 is _8
(double the
difference).
```

Conjunction

```
     f =: -@:%                    NB. same as & in monadic form
     f 1 2 3 4 5
_1 _0.5 _0.333333 _0.25 _0.2

     10 f 1 2 3 4 5               NB. i.e., - (x % y)
_10 _5 _3.33333 _2.5 _2

     f =: +/@:%                   NB. i.e., +/ (% y)
     f 1 2 3 4 5                  NB. i.e., sum reciprocals of y
2.28333
```

Challenge!

Define the L^1, L^2, and L^∞ norms.

L^1 = sum(absval(x_i)) for all x_i

L^2 = sqrt(sum(x_i^2) for all x_i)

L^∞ = max(x_i) for all x_i

Find <u>one</u> solution for <u>each</u> norm using all of these cards simultaneously:

2x `+` 3x `m/` `u/` 3x `u@:v`

1x `|` 1x `*:` 1x `%:` 1x `>.`

Atoms, Lists, Tables, et al.

,

, — — —

Ravel

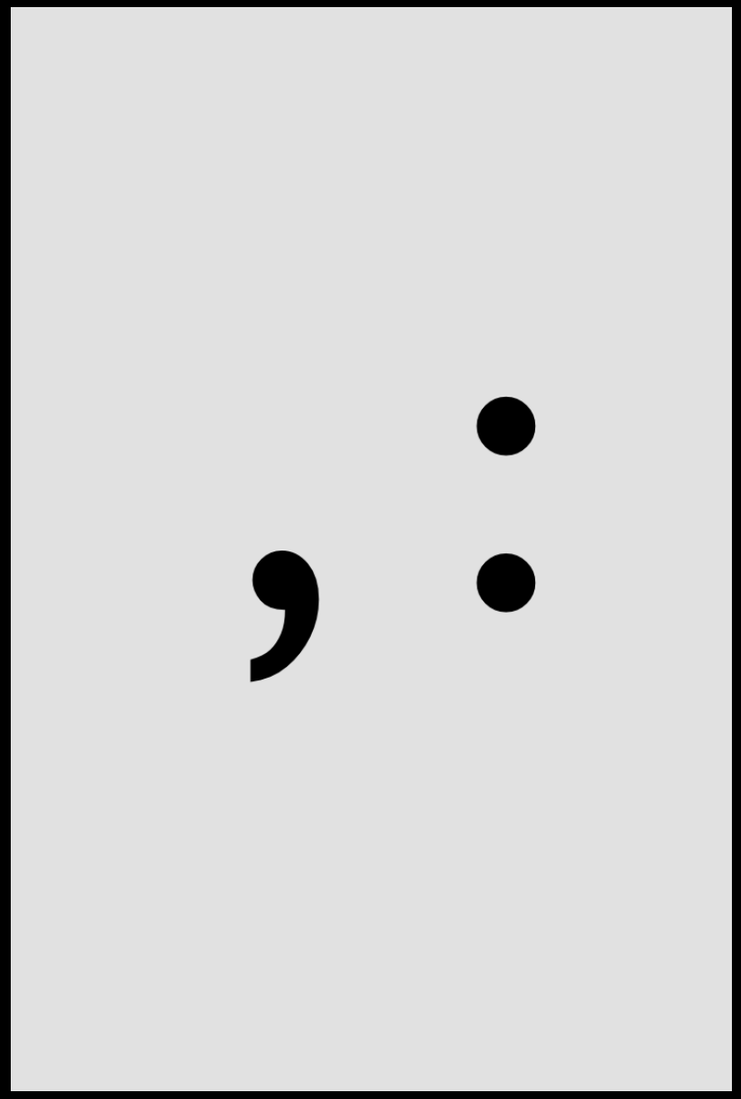
,y gives a list of the atoms of y in “normal” order: the result is ordered by items, by items within items, etc. The result shape is 1$*/$y .

Append

x,y appends items of y to items of x .

Verb

```
    (i.3) (i.5)
|syntax error
|       (i.3)(i.5)
    (i.3),(i.5)
0 1 2 0 1 2 3 4
```



,: — — —

Itemize

,:y adds a leading unit axis to y , giving a result of shape 1,$y .

Laminate

An atomic argument in x,:y is first reshaped to the shape of the other (or to a list if the other argument is also atomic); the results are then itemized and catenated, as in (,:x),(,:y) .

Verb

```
      pt1 =: 5.1 3.5 1.4 0.2
      pt2 =: 4.9 3.0 1.4 0.2
      pt3 =: 4.7 3.2 1.3 0.2
      pt1 , pt2 , pt3
5.1 3.5 1.4 0.2 4.9 3 1.4 0.2 4.7 3.2 1.3 0.2
      pt1 , pt2 ,: pt3
5.1 3.5 1.4 0.2
4.9   3 1.4 0.2
4.7 3.2 1.3 0.2
```

$

$ _ 1 _

Shape Of

$ y yields the shape of y . For example, the shape of a 2-by-3 matrix is 2 3, and the shape of the scalar 3 is an empty list (whose shape is 0). The rank of an argument y is #@$ y .

Shape

The shape of x $y is x,siy where siy is the shape of an item of y; x$y gives a length error if y is empty and x,siy does not contain a zero.

Verb

```
     pt1 , pt2 , pt3
5.1 3.5 1.4 0.2 4.9 3 1.4 0.2 4.7 3.2 1.3 0.2
     $ pt1 , pt2 , pt3
12
     pt1 , pt2 ,: pt3
5.1 3.5 1.4 0.2
4.9   3 1.4 0.2
4.7 3.2 1.3 0.2
     $ pt1 , pt2 ,: pt3
3 4
```

Rank

The rank of a noun is the count of its axes. An atom has rank 0, a list rank 1, a table rank 2, and an array with 5 axes has rank 5.

J Primer

```
    $ pt1 , pt2 , pt3
12
    # $ pt1 , pt2 , pt3  NB. #$ computes rank
1
    $ pt1 , pt2 ,: pt3
3 4
    # $ pt1 , pt2 ,: pt3
2
    NB. etc.
```

Ask the rank of a verb.

```
     + b.0
0 0 0
     - b.0
0 0 0
     i. b.0
1 _ _
     $ b.0
_ 1 _
     ,: b.0
_ _ _
```

If you don’t know the rank of a verb, you don’t know the verb. Using a verb of unknown rank is like wiring in a power-supply of unknown voltage–it will do something when you plug it in; it might even work; but if the voltage is wrong it will destroy what it’s connected to. Avoid embarrassment! Know the rank of the verbs you use.

J Primer

u"n

u"n

Rank

```
The verb u"n applies u to each
cell as specified by the rank n .
The full form of the rank used is
3$&.|.n . For example, if n=:2 ,
the three ranks are 2 2 2 , and
if n=: 2 3, they are 3 2 3 . A
negative rank is complementary:
u"(-r) y is equivalent to u"(0>.
(#$y)-r)"_ y .
```

Conjunction

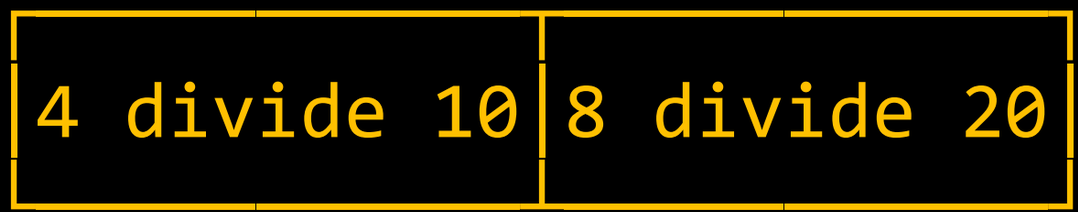
```
     100 200 300 + (1 2 3 4)
|length error
|    100 200 300    +(1 2 3 4)

     100 200 300 +"0 _ (1 2 3 4)
101 102 103 104
201 202 203 204
301 302 303 304

     100 200 300 +"_ 0 (1 2 3 4)
101 201 301
102 202 302
103 203 303
104 204 304
```

Show how a verb would be applied.

```
load 'general/misc/fndisplay'
defverbs 'divide'
setfnform 'J'
8 divide 10
```


```
4 8 divide 10 20
```

4 8 (divide"0) 10 20

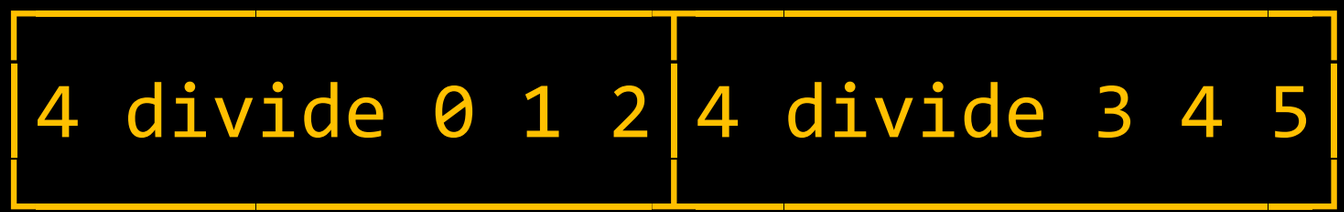


4 8 (divide"1) 10 20

```
     i.2 3
0 1 2
3 4 5
     4 (divide"0) i.2 3
```

4 divide 0	4 divide 1	4 divide 2
4 divide 3	4 divide 4	4 divide 5

```
   4 (divide"1) i.2 3
```




```
   4 (divide"_) i.2 3
```

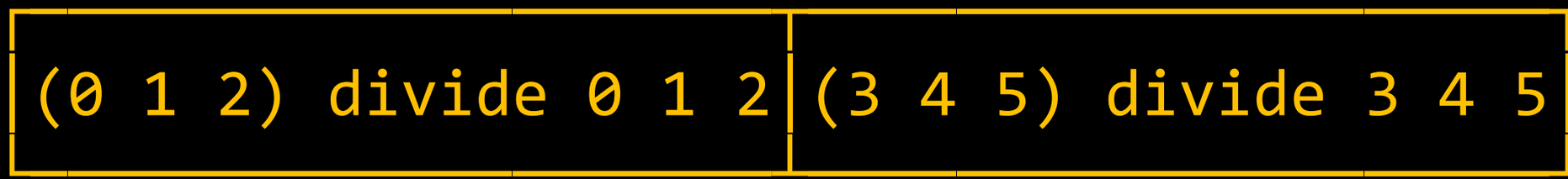
```
(i.2 3) (divide"0 0) i.2 3
```

0 divide 0	1 divide 1	2 divide 2
3 divide 3	4 divide 4	5 divide 5

```
   (i.2 3) (divide"1 0) i.2 3
```

(0 1 2) divide 0	(0 1 2) divide 1	(0 1 2) divide 2
(3 4 5) divide 3	(3 4 5) divide 4	(3 4 5) divide 5

```
(i.2 3) (divide"1 1) i.2 3
```



```
      (i.2 3) (divide"1 _) i.2 3
```

(0 1 2) divide 2 3$(0 1 2) (3 4 5)	(3 4 5) divide 2 3$(0 1 2) (3 4 5)

```
      (i.2 3) (divide"_ 0) i.2 3
```

(2 3$(0 1 2) (3 4 5)) divide 0	(2 3$(0 1 2) (3 4 5)) divide 1	(2 3$(0 1 2) (3 4 5)) divide 2
(2 3$(0 1 2) (3 4 5)) divide 3	(2 3$(0 1 2) (3 4 5)) divide 4	(2 3$(0 1 2) (3 4 5)) divide 5

```
      (i.2 3) (divide"0 _) i.2 3
```

0 divide 2 3$(0 1 2) (3 4 5)	1 divide 2 3$(0 1 2) (3 4 5)	2 divide 2 3$(0 1 2) (3 4 5)
3 divide 2 3$(0 1 2) (3 4 5)	4 divide 2 3$(0 1 2) (3 4 5)	5 divide 2 3$(0 1 2) (3 4 5)

```
      (i.2 3) (divide"_ _) i.2 3
```

(2 3$(0 1 2) (3 4 5)) divide 2 3$(0 1 2) (3 4 5)

In J, every operator has a loop built in.

J for C Programmers

Coding Challenge!

Write code that fills in the blanks and produces the given output.

Compute the column sums of a table.

```
      _______ i.4 6
36 40 44 48 52 56
```

Compute the arithmetic mean of every row in a pair of tables.
Recall the arithmetic mean is +/%#

```
      _______ i.2 3 6
 2.5   8.5 14.5
20.5 26.5 32.5
```

Recursion

u^:n
u^:v

u^:n u^:v

Power

The verb u is applied n times. An infinite power n produces the limit of the application of u . If n is negative, the obverse u^:_1 is applied |n times.

```
x u^:n y ==
x&u^:n y
```

The case of ^: with a verb right argument is defined in terms of the noun right argument case (u ^: n) as follows:

```
u ^: v y ==
u^:(v y) y ;
x u ^: v y ==
x u^:(x v y) y
```

Conjunction

Recursive “Larger of” >.

```
     vals=:4 3 2 $ 24?100
     >./vals
96 95
70 85
60 89
     >./>./vals
96 95
     >./>./>./vals
96
     >./^:3 vals NB. recursive application, 3 times
96
     >./^:_ vals NB. 'limit' of >./, works on any shape
96
```

```
      NB. interesting cases of u^:n

      NB. do-once
      (>:^:1) 10          NB. increment once
11
      NB. do-many
      (>:^:5) 10          NB. increment five times
15
      NB. don’t
      (>:^:0) 10          NB. returns y
10
```

```
      NB. interesting cases of u^:n

      NB. do various times
      (>:^:(i.10)) 10    NB. increment 0, 1, …, 9 times
10 11 12 13 14 15 16 17 18 19

      NB. obverse (aka inverse)
      (>:^:_1) 10        NB. decrement once
9

      NB. do-infinitely-many-times
      (cos^:_) 1         NB. compute y=cos(y)
0.739085
```

u^:n
u^:v

u^:n u^:v

Power

The verb u is applied n times. An infinite power n produces the limit of the application of u . If n is negative, the obverse u^:_1 is applied |n times.

```
x u^:n y ==
x&u^:n y
```

The case of ^: with a verb right argument is defined in terms of the noun right argument case (u ^: n) as follows:

```
u ^: v y ==
u^:(v y) y ;
x u ^: v y ==
x u^:(x v y) y
```

Conjunction

```
      NB. highly interesting cases of u^:v

      NB. do sometimes
      1 2 3 4 5 ((>:@[)^:])"0 (1 0 0 1 0)
2 2 3 5 5

      NB. explanation:
      NB. ] keeps right side, [ keeps left side
      NB. >:@[ means "increment vals of left argument"
      NB. ^:] means apply power (^:) n times, where n
      NB.      comes from right side
      NB. finally, "0 means apply ^: to each value
```

```
      NB. written another way…

      myfunc =: >:  NB. define function to apply
      if =: adverb define
''                  NB. no monadic form
:
y u@]^:["0 x        NB. here is the dyadic form
)

      1 2 3 4 5 myfunc if 1 0 0 1 0
2 2 3 5 5
```

```
      NB. more fancy

      even =: monad : '0=2|y'   NB. checks that y is even
      myfunc =: >:

      NB. use a hook!
      (myfunc if even) 1 2 3 4 5
1 3 3 5 5

      NB. the above turned into:
      NB. 1 2 3 4 5 (myfunc if) (even 1 2 3 4 5)
      NB. 1 2 3 4 5 (myfunc if) 0 1 0 1 0
```

```
     NB. do while…

     NB. (u^:v)^:_ ends when the output of v is the same
     NB. as the prior time; supposing v is boolean (0/1),
     NB. then ^:_ stops when v produces 0, yielding u^:0

     dowhile =: conjunction define
(u^:(v@:])^:_"0) y
:
x (u^:(v@:])^:_"0) y
)
     NB. continue doubling a number until it’s >= 100
     2 * dowhile (100&>) 1 3 5 7 9 11
128 192 160 112 144 176
```

```
      NB. defining “if” and “dowhile” as a J adverb (if)
      NB. and conjunction (dowhile) gives us the benefits
      NB. of automatic looping:

      (2&* if even) i. 2 3 2
 0  1
 4  3
 8  5

12  7
16  9
20 11
```

```
      NB. defining “if” and “dowhile” as a J adverb (if)
      NB. and conjunction (dowhile) gives us the benefits
      NB. of automatic looping:

      (2&* dowhile (100&>)) i. 2 3 2
  0 128
128 192
128 160

192 112
128 144
160 176
```

Coding Challenge!

Write code that fills in the blanks and produces the given output.

Given a number N and a list of scores, compute the average of the scores after dropping the lowest N scores. Create one solution using `^:` and one solution without using `^:` Feel free to use define.

```
     1 ______ 90 70 80 75          NB. drop 1 score
81.6667
     2 ______ 90 70 80 75          NB. drop 2 scores
85
     3 ______ 95 60 0 55 80 0 90   NB. drop 3 scores
78.333
```

Big Data

You may at first worry that you're using too much memory, or that you might misuse the processor's caches; get over it. Apply verbs to large operands.

J for C Programmers

Jd, the J relational database.

```
load 'jd'

NB. create a database folder
jdadminx'bbjd'

NB. set path to CSV files to load
CSVFOLDER=:F=:'/home/jeckroth/chautauquas/zigzag/'

NB. create a "cdefs" file that describes the dataset
jd'csvcdefs /h 1 backblaze-2013-2016.csv'
```

```
NB. load the backblaze data (60mil rows)
jd'csvrd backblaze-2013-2016.csv backblaze'

NB. the data are stored as one binary file per column;
NB. when you come back, just type jdadmin'mydbname'

NB. show the db schema
jd'info schema'
```

table	column	type	shape
backblaze	date	edate	_
backblaze	serial_number	byte	15
backblaze	model	byte	23
backblaze	capacity_bytes	int	_
backblaze	failure	boolean	_
backblaze	smart_187_raw	int	_

```
      NB. read some data
      jd'reads date,capacity_bytes from backblaze'
...
┌──────────┬──────────────┐
│2013-04-10│  3000592982016│
│2013-04-10│  3000592982016│
│2013-04-10│  3000592982016│
│2013-04-10│  2000398934016│
│2013-04-10│  2000398934016│
...
│2016-12-29│3220561902732421│12
│2016-12-30│3220521894862120│96
│2016-12-31│3220561902732421│12
└──────────┴──────────────┘

      NB. save result
      capacity_by_day =: (...above...)
```

```
      NB. results are "boxed" (each box can hold a different type)
      NB. so let's unbox just the capacities
      capacities =. ,/ > 1  1 { capacity_by_day
      capacities
5404121422221312 0 5404121422221312 0 5404121422221312 0
5386117880586240 0 5565753218233958 4 5565753218233958 4
5579255837881958 4 5579655916584960 0 5579655916584960 0
5579655916584960 0 5579655916584960 0 5579505886393958 4
5579505886393958 4 5593008554813030 4 ...
      $ capacities
1360
      NB. find min, max capacity
      <./ capacities
_8946789308003949568
      >./ capacities
12824295941861194 24
```

NB. avg capacity
(+/%#) capacities
3.55142e12

NB. or ask jd to compute avg using one of several agg verbs
jd'info agg'

aggs
avg
count
countunique
first
last
max
min
sum

jd'reads avg capacity_bytes from backblaze

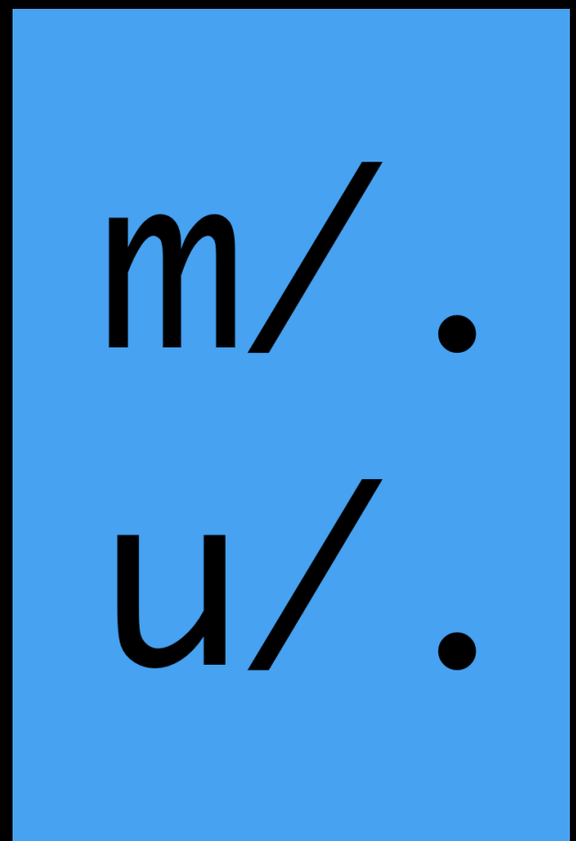
capacity_bytes
3.55142e12

NB. 5sec

```
     NB. compute overall failure rate
     failures =: ,/>0{1{ jd'reads failure from backblaze'
     $ failures
59654783
     +/failures          NB. total failures
5806
     overallAFR =: 365*100*(+/failures)%(#failures)
     overallAFR
3.55242
```

```
     NB. compute failures by year
     dates =: >0{1{ jd'reads date from backblaze'
     $ dates
59654783 10          NB. ugh, dates are strings now
     2 {. dates
2013-04-10
2013-04-10
     (4&{."1) 2 {. dates  NB. keep only first four digits
2013
2013
     NB. 0". converts strings to numbers
     years =: 0 ". (4&{."1) dates
     $ years
59654783
```

```
NB. ok, we have years
NB. how do we sum failures per year?

NB. note /. adverb, known as "key"
```



m/. u/. _ _ _

Oblique

u/.y applies u to each of the oblique lines of a table y . m/.y applies successive verbs from the gerund m to the oblique lines of _2-cells of y, extending m cyclically as required.

Key

x u/.y == (=x) u@# y , that is, items of x specify keys for corresponding items of y and u is applied to each collection of y having identical keys. x m/.y applies successive verbs from the gerund m to the collections of y, extending m cyclically as required.

Adverb

```
      NB. ok, we have years
      NB. how do we sum failures per year?

      NB. note /. adverb, known as "key"

      NB. example usage:
      2013 2013 2014 2015 2015 +//. 2 3 7 11 12
5 7 23

      NB. hence:
      driveFailuresYears =: years +//. failures   NB. 1sec
      driveFailuresYears
740 2206 1429 1431
```

```
      NB. likewise:
      driveDays =: years #/. failures
      driveDays
5091501 12582414 17509251 24471617

      NB. get unique years (in order of data) to match
      NB. the driveFailuresYears result
      uniqueYears =: ~.years           NB. 1sec
      uniqueYears
2013 2014 2015 2016

      NB. hence hence:
      yearlyAFR =: 365*100*driveFailuresYears%driveDays
      uniqueYears ,: yearlyAFR
   2013    2014    2015    2016
5.30492 6.39933 2.97891 2.13437
```

The big slide with timing data.

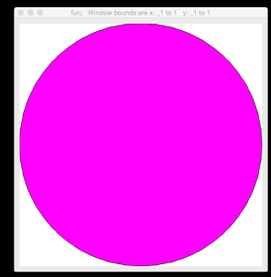
Task	Jd	R (dplyr)	MySQL
Import CSV data	50 sec	2 min (read_csv)	11 min
Space on disk	3.6 GB	374 MB	5.2 GB
Load data into variable	55 sec	1 min 45 sec	NA
Select sum(capacity) group by date	~1 sec	~1 sec	4.5 min
Select avg(capacity)	~1 sec	~1 sec	38 sec

Note, original CSV data was 3.2 GB.

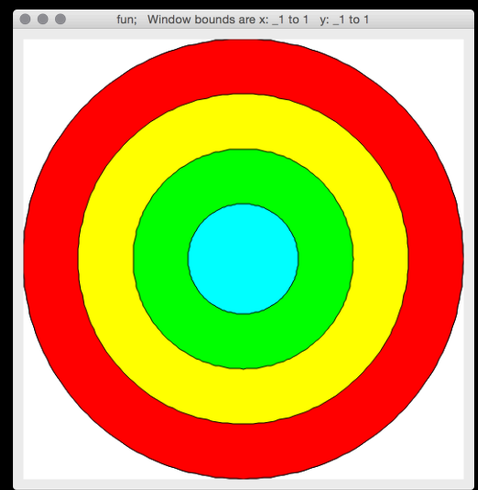
Visualization

```
      load '~addons/graphics/fvj4/dwin.ijs'
      _1 _1 1 1 dwin 'fun'    NB. open window, -1:1 axes
      1 o. 1r6p1              NB. sin(1/6 pi^1)
0.5
      2 o. 1r6p1              NB. cos(1/6 pi^1)
0.866025

      NB. define x,y coordinates of a circle with 200 pts
      circ =: |: 2 1 o./ 1r100p1*i.200
1           0
0.999507    0.0314108
0.998027    0.0627905
0.995562    0.0941083
...
```

```
NB. plot a circle with a color
255 0 255 dpoly circ
```



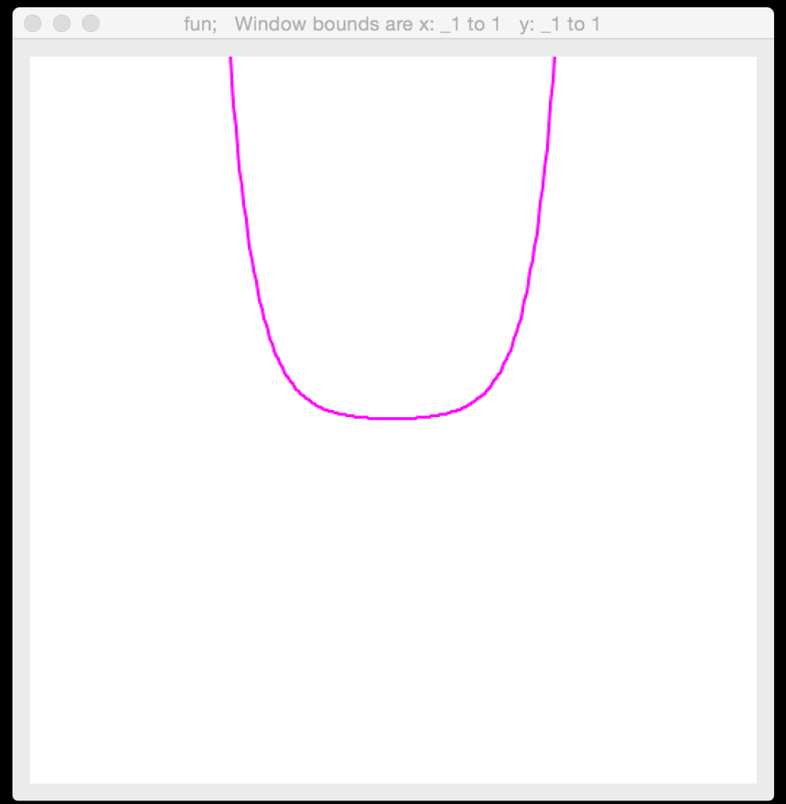
```
circ4 =: 1 0.75 0.5 0.25 */ circ  NB. four circles
colr4 =: 255 0 0, 255 255 0, 0 255 0,: 0 255 255

NB. plot four circles with individual colors
colr4 dpoly circ4
```



```
      NB. generate some exponential values
      (^ 0.1 * i.80) % 100
0.01 0.0110517 0.012214 0.0134986 ...

      NB. generate some x,y values for a symmetric curve
      xs =: ((i.180) * %180) - 0.5
      ys =: ((|. (^0.075*i.90)), (^0.075*i.90))%400
      curve =: |: xs ,: ys
      curve
        _0.5    1.98087
  _0.494444    1.83774
  _0.488889    1.70495
...
```

```
      curve
         _0.5     1.98087
   _0.494444     1.83774
   _0.488889     1.70495
...
  _0.0111111 0.00269471
_0.00555556      0.0025
           0      0.0025
 0.00555556 0.00269471
  0.0111111 0.00290459
...
    0.483333     1.70495
    0.488889     1.83774
    0.494444     1.98087
```

NB. plot the curve
255 0 255 dline curve



Rotation can be defined as matrix multiplication of a vector representing the point: [x,y,1] and a rotation matrix:

```
M = [ cos(θ) -sin(θ) 0
      sin(θ)  cos(θ) 0
           0       0 1 ]
```

After multiplying M*[x,y,1]^{T}, the result will be [x',y',z]; divide z out of x' and y' to get final coordinates. In our case, however, z will always be 1.

```
NB. define matrix mult.
mp =: +/ . *

NB. allow simple trig names
load 'trig'

NB. define rotation matrix
rotm =: (cos, sin, 0:),
        (-@sin, cos, 0:),:
        (0:, 0:, 1:)
rotm 1r4p1
 0.707107 0.707107 0
_0.707107 0.707107 0
        0        0 1
```

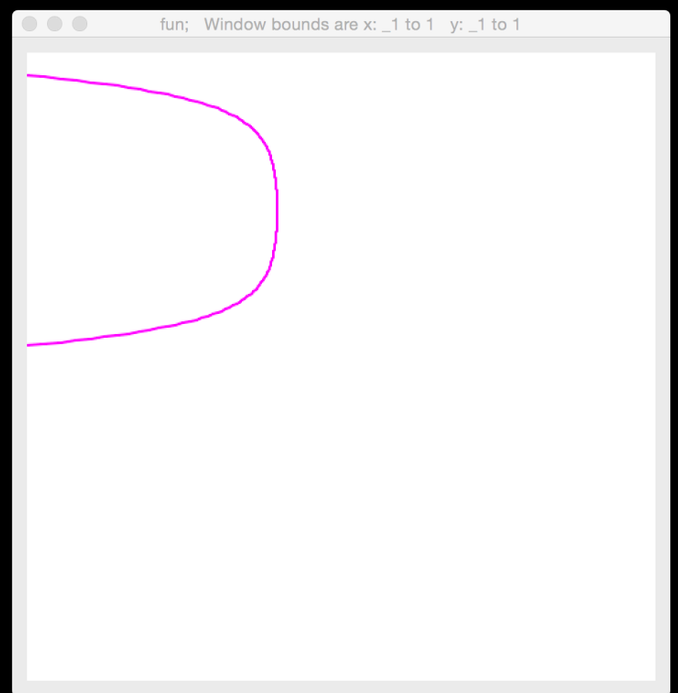
Likewise, a translation matrix is defined as:

```
M = [ 1  0  xtrans
      0  1  ytrans
      0  0  1      ]
```

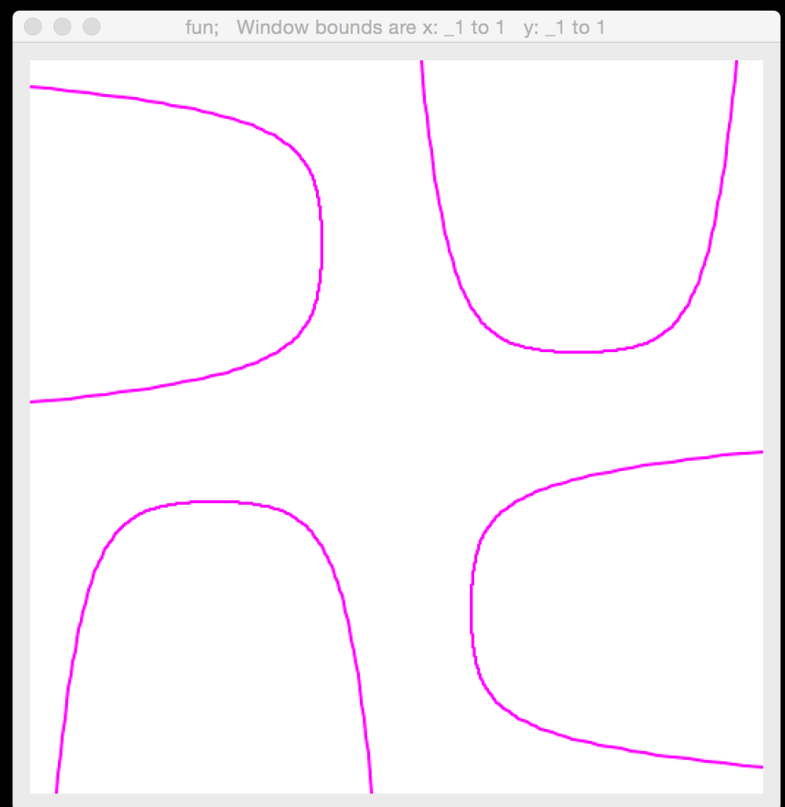
Translation & rotation matrices can be combined with matrix multiplication, e.g.,
TransM * RotM * $[x,y,1]^T$

```
     NB. trans. matrix as verb
     trans =: monad define
1 0 0, 0 1 0,: (y,1)
)
     trans 5 2
1 0 0
0 1 0
5 2 1
     (trans 5 2) mp (rotm 1r4p1)
 0.707107 0.707107 0
_0.707107 0.707107 0
  2.12132  4.94975 1
     (3 2 1) mp (... above ...)
2.82843 8.48528 1
```

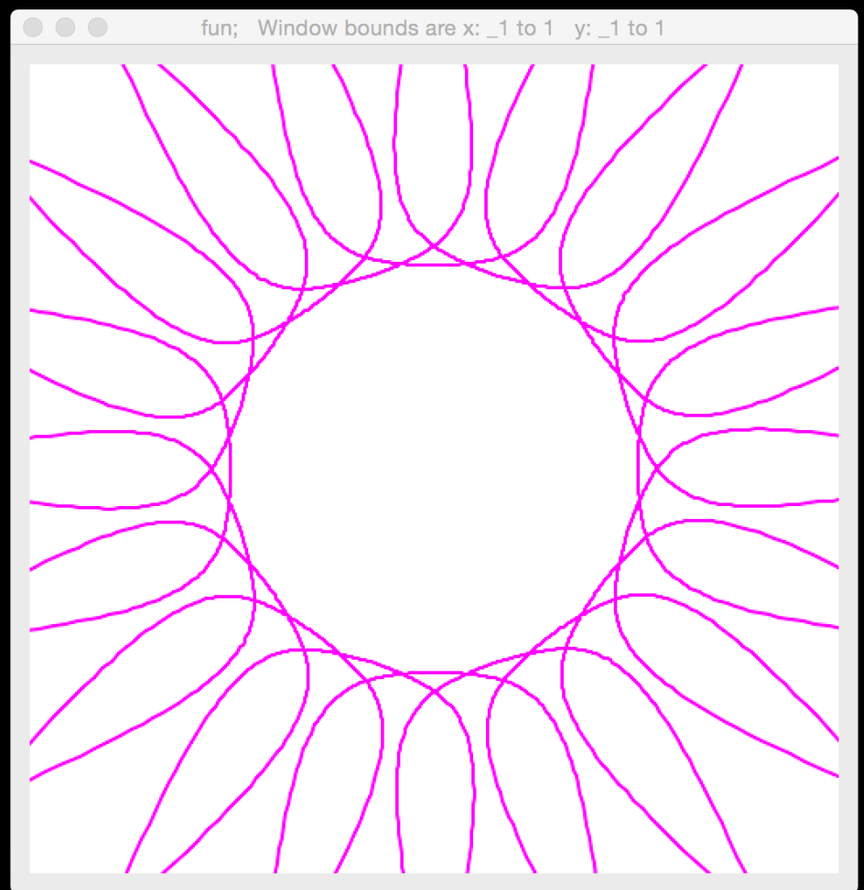
```
NB. add extra z=1 dimension to curve
curve =: curve,.1

NB. draw a translated, rotated curve
dclear''
255 0 255 dline curve mp (trans 0.5 0.2) mp (rotm 1r2p1)
```



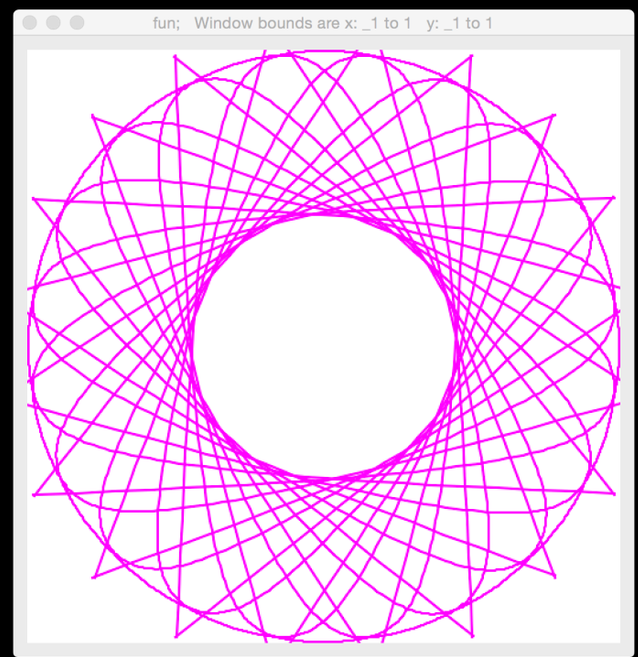
```
      NB. define a verb for drawing curve with translate + rotate
      NB. use dyad form, x=translate, y=rotate
      dcurveRot =: dyad define
255 0 255 dline curve mp (trans x) mp (rotm y)
)
      dclear''
      0.5 0.2 dcurveRot 1r2p1
      0.5 0.2 dcurveRot 2r2p1
      0.5 0.2 dcurveRot 3r2p1
      0.5 0.2 dcurveRot 4r2p1
```



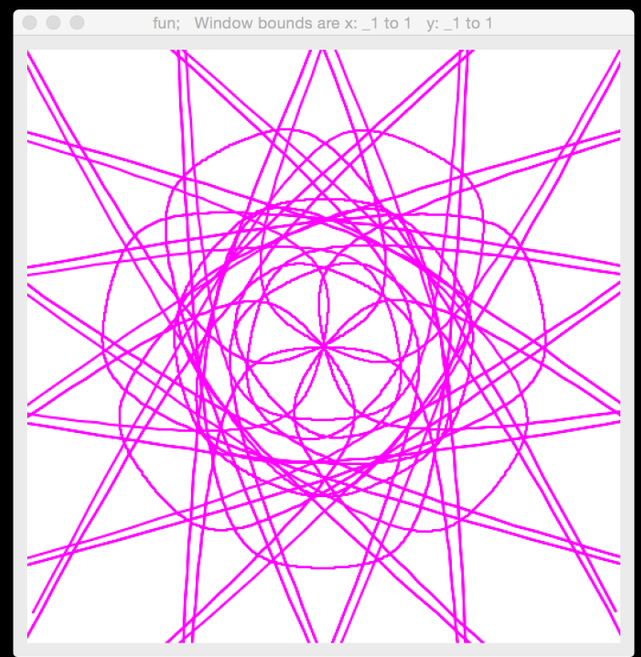
```
NB. draw many!
dclear''
(0 0.5 & dcurveRot)"0 (1r10p1 * i.20)
```



```
      NB. unrotate by pi as the last step before drawing,
      NB. and eliminate x-coordinate of translation parameter
      dcurveRot =: dyad define
255 0 255 dline curve mp (rotm _1p1) mp (trans 0,x) mp (rotm y)
)
      dclear''
      1 dcurveRot"0 (1r10p1 * i.20)
```



```
NB. generate 4 translations 0.25 apart
translations =: 0.25 * i.4
NB. generate 14 rotations 1/7 pi apart
rotations =: 1r7p1 * i.14
NB. draw each rotation/translation pairing (48 pairings)
dclear''
(48$translations) dcurveRot"0 (48$rotations)
```

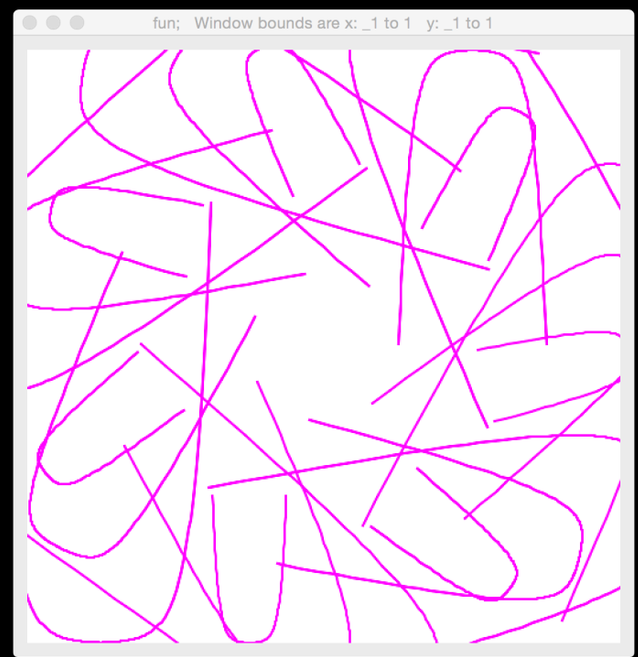


A scaling (zooming) matrix can be defined as:

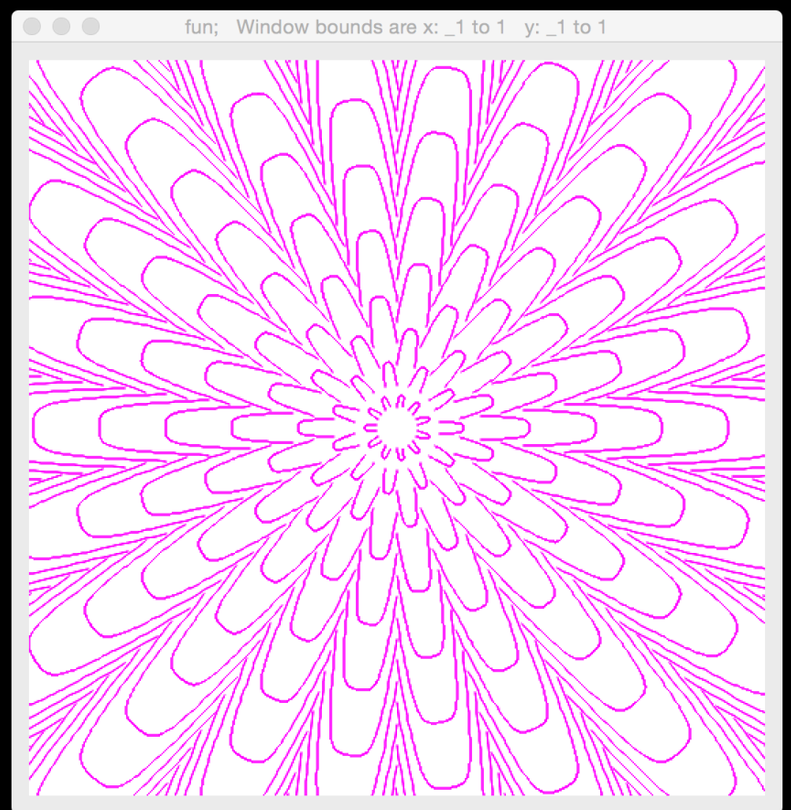
```
M = [ s  0  0
      0  s  0
      0  0  1 ]
```

```
      NB. scaling matrix as verb
      scale =: monad define
(y, 0 0), (0, y, 0),: 0 0 1
)
      scale 5
5 0 0
0 5 0
0 0 1
      (scale 5) mp (trans 5 2) mp (rotm 1r4p1)
 3.53553 3.53553 0
_3.53553 3.53553 0
 2.12132 4.94975 1
      (3 2 1) mp (... above ...)
5.65685 22.6274 1
```

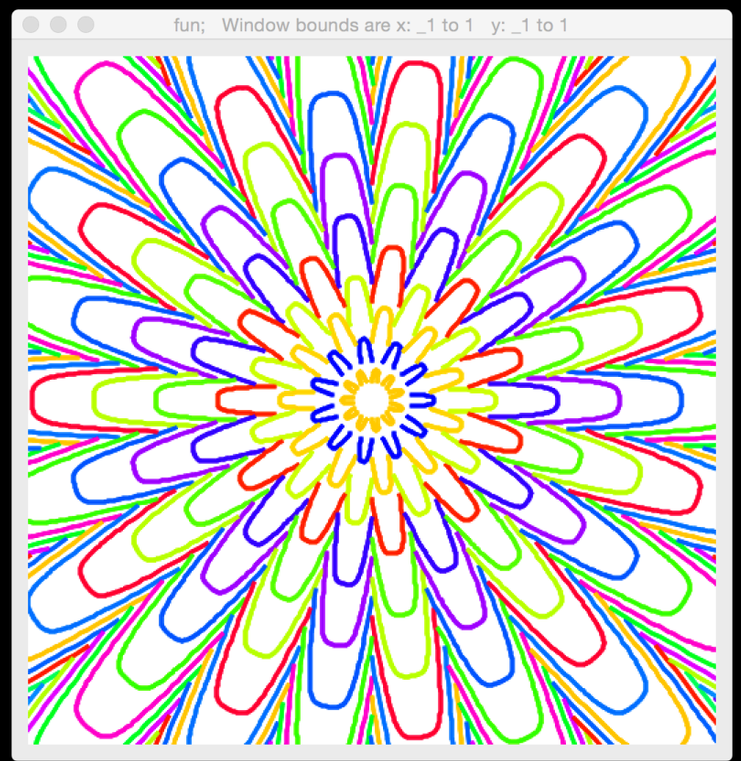
```
      NB. update dcurveRot to use scale based on translation size
      dcurveRot =: dyad define
m =. (scale x) mp (rotm _1p1) mp (trans x) mp (rotm y)
255 0 255 dline curve mp m
)
      dclear''
      (48$translations) dcurveRot"0 (48$rotations)
```



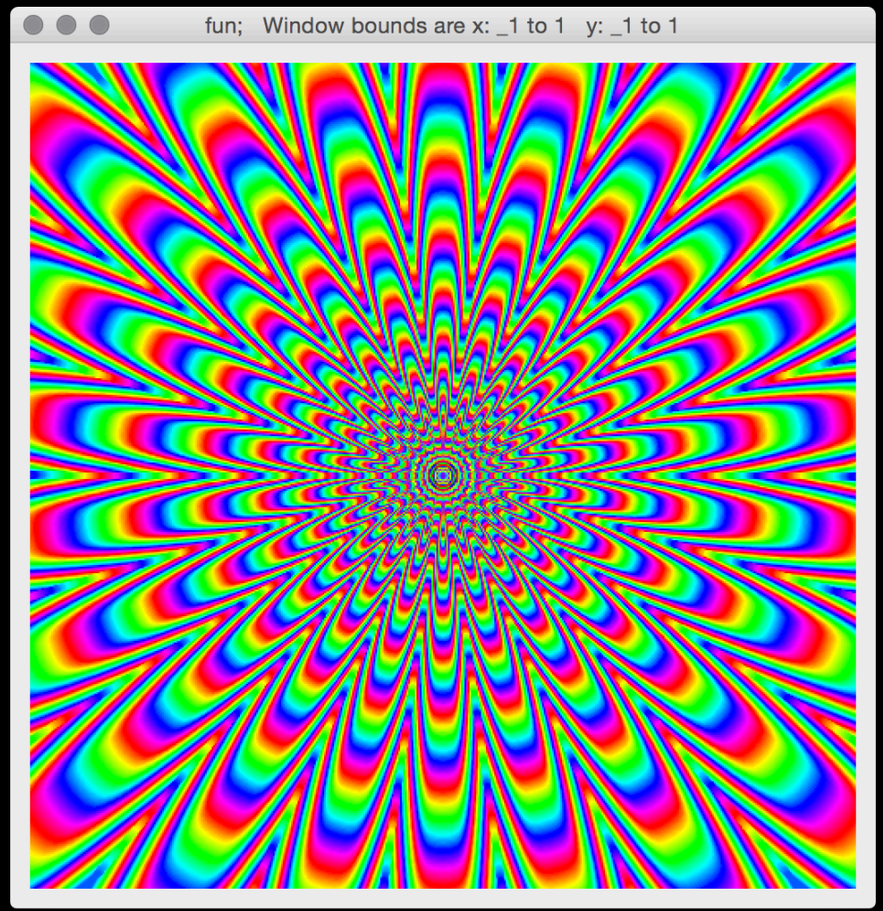
```
NB. update dcurveRot to use (scale 0.185*x) and
NB. obsessively fine-tune numbers to arrive at:
translations =: 0.09*i.50
rotations =: 1r22p1+1r11p1*i.22
(1100$translations) dcurveRot"0 (1100$rotations)
```



```
        NB. next we need to set the color based on x;
        NB. we'll use HSV scale for gradual color changes

        dcurveRot =: dyad define
m =. (scale 0.185*x) mp (rotm _1p1) mp (trans 0,x) mp (rotm y)
(hsv2rgb ((360|90000*x^0.015),1.0 1.0 1.0)) dline curve mp m
)
```



```
NB. MOAR CURVES
translations =: 0.005 * i.400
rotations =: 1r11p1*i.22
(22#translations) dcurveRot"0 (8800$rotations)
```



NB. add those extra curves rotated pi/22 and call it a day