

J Phrases



Chris Burke

Roger K. W. Hui

Kenneth E. Iverson

Eugene E. McDonnell

Donald B. McIntyre

Copyright © 1991-2002 Jsoftware Inc. All Rights Reserved.

Last updated: 2002-11-08

www.jsoftware.com

Table of Contents

1	<u>A. Conventions</u>
2	<u>B. Loading Scripts</u>
3	<u>C. Specimens</u>
4	<u>D. Function Tables</u>
5	<u>A. Bond or Curry</u>
6	<u>B. Rank</u>
7	<u>C. Ambivalent Functions</u>
8	<u>D. Adverbs & Conjunctions</u>
9	<u>E. Explicit Definitions</u>
10	<u>F. Hooks</u>
11	<u>G. Verb Trains</u>
12	<u>H. Gerunds</u>
13	<u>A. Indexing</u>
14	<u>B. Merge & Amend</u>
15	<u>A. Intervals</u>
16	<u>B. Locating & Selecting</u>
17	<u>C. Test</u>
18	<u>A. Structural</u>
19	<u>B. Partition</u>
20	<u>C. Special Matrices & Lists</u>
21	<u>A. Sorting</u>
22	<u>B. Ranking & Classification</u>
23	<u>C. Grading</u>
24	<u>A. Permutations</u>
25	<u>B. Rotations & Reflections</u>
26	<u>C. Parity & Symmetry</u>
27	<u>A. Numbers & Counting</u>
28	<u>B. Grids</u>
29	<u>C. Representations</u>
30	<u>D. Arithmetic</u>
31	<u>E. Complex Numbers</u>

32	<u>A. Matrix Algebra</u>
33	<u>B. Linear Vector Functions</u>
34	<u>C. Polynomials & Rational Functions</u>
35	<u>D. Transcendental Functions</u>
36	<u>E. Quadrature & Simpson's Rule</u>
37	<u>F. Geometry</u>
38	<u>A. Sums & Means</u>
39	<u>B. E Pluribus Unum</u>
40	<u>C. Math & Stats</u>
41	<u>D. Plotting</u>
42	<u>E. Approximation</u>
43	<u>F. Random Numbers</u>
44	<u>A. Inverse</u>
45	<u>B. Duality</u>
46	<u>C. Transformations</u>
47	<u>A. Finance</u>
48	<u>A. Inside Boxes</u>
49	<u>B. Character</u>
50	<u>C. Type Change</u>
51	<u>A. Execution Time & Space</u>
52	<u>B. Date & Time</u>
53	<u>A. Case Statements</u>
54	<u>B. Miscellaneous</u>
55	<u>REFERENCES</u>
56	<u>Index</u>

1A. Conventions

J is an executable mathematical language available on a wide range of computers. This book presents a collection of J phrases, useful to beginners in learning the language, and of continuing use to practical programmers.

Each phrase is executable and includes assignment to a name whose numeric part is unique within the section. The literal part of the name indicates the class of the phrase as follows:

a	adverb	m	monad
c	conjunction	n	noun
d	dyad	v	ambivalent verb

Some phrases include assignment of a mnemonic name that is not necessarily unique. Each is accompanied by a brief statement of its purpose. Thus:

n0=: i. 6	List of first six non-negative integers
m1=: ^&3	Cube
m2=: mean=: +/ % #	Arithmetic mean
d3=: \$, :	x copies of y
m4=: <.@(0.5&+)	Round
m5=: =+	Test for real number
m6=: (<0 _1)&C.	Swap leading and final items
m7=: +/\ -: +/\.&. .	Prefix sum scan is suffix under reversal

A beginner may find it useful to simply browse through, and enter, various portions of the phrase lists. In so doing, she may best skip those whose purpose is described by unfamiliar terms, and concentrate first on those phrases described by words that are familiar, or at least to be found in an English dictionary.

Complete definitions of the primitives used (such as `&` in `m1`) may be found in [\[5\]](#) [p55](#), but it is probably more fruitful and more fun to first experiment by entering a

phrase together with a suitable argument to see it in action, and then experiment further by entering what may appear to be meaningful variants. You may also wish to annotate these experiments (but do not enter the annotation on the computer unless you preface it by NB.). For example:

```

n0=: i. 6
n0 NB. Enter name of noun to display it
0 1 2 3 4 5

m1=:      n0=: i. 6
n0 NB. Enter name of noun to display it
0 1 2 3 4 5

m1=: ^&3
m1 n0
0 1 8 27 64 125

m2=: mean=: +/ % #
m2 n0
2.5

mean m1 n0
37.53
m1 n0
0 1 8 27 64 125

```

Experimenting with an unfamiliar language is bound to lead to some puzzling results:

```

^&3 y=: 0 1 2 3 4 NB. Power with 3 is the cube
0 1 8 27 64

^&3 0 1 2 3 4 NB. A puzzling result
+-----+
|^|&|3 0 1 2 3 4|
+-----+

^&3 (0 1 2 3 4) NB. The list must be isolated from the 3 of
the phrase
0 1 8 27 64 NB. so that it does not attach to it to form a
longer list

```

Spend a little time trying to understand such problems, but do not let them deter you from further experiments or detain you very long; it is better to forge ahead and return to them, or perhaps forget them as they will probably clear up in the course of further work.

In experimenting with phrases, begin with the simplest arguments that occur to you, but progress to more complex ones to explore the potential of the phrase. For example, `m4` appears useless when applied to an integer, useful when applied to a fraction, and perhaps more interesting when applied to a negative fraction. Similarly, `m5` is uninteresting when applied to real numbers, but it (as well as the primitive `+`) is useful on complex numbers such as `2j3` and `%:_1` (the square root of negative one).

Phrases such as `m2` and `m6` begin to show interesting results only when applied to lists, such as in `m6 'ABCDabcd'`. They should also be tried on tables (matrices) and reports (higher-rank arrays) such as `i. 3 4` and `i.3 4 5` and `2 3 4$'ABCDEFG'`. Also try expressions such as `m6"2 i.3 4 5`. For example:

```

report=: ? 2 4 3 $ 10 Report for two years of four
quarters of three months
report
1 7 4
5 2 0
6 6 9
3 5 8

0 0 5
6 0 3
0 4 6
5 9 8

mean report
0.5 3.5 4.5
5.5 1 1.5
3 5 7.5
4 7 8

(mean ; mean"2 ; mean"1) report
+-----+
|0.5 3.5 4.5|3.75 5 5.25| 4 2.33333 7 5.33333|

```

5.5	1	1.5	2.75	3.25	5.5	1.66667	3	3.33333	7.33333
3	5	7.5							
4	7	8							

Parameters that occur in phrases (such as the 3 in `m1`) invite further experiments. Substitute both positive and negative integers for the parameter in `m1`; experiment to determine the range of permissible values, and try to state in English the general definition of the primitive `^`. Also see the relevant section on permutations.

Although the `d` in the name of the phrase `d3` indicated that it was for dyadic use only, it will also be found to give a (perhaps unintelligible) result on a single argument as well. For example, try `3 d3 2 3` and `d3 2 3`. Thus the `d` in the name of the phrase merely suggests its intended use, and does nothing to prevent its use as a monad. However, a verb so restricted can be defined by `f=: [: : d3`.

The definition of a verb, adverb, or conjunction may be displayed in any of three forms: box, tree, or linear (the form mainly used herein). The form may be chosen from the view menu, or by the foreign conjunction `9! : 3` as illustrated below:

```
9! : 3 (5) Linear display
m7
+/\ -: +/\.&.|.
```

```
9! : 3 (2 4 5) Box, tree, and linear display
m7
+-----+
|+-----+|-: |+-----+| | | | | | | | | | | |
||+---+|\|| | |+-----+|&.| |.||
|||+|/|| | | ||+---+|\.| | | |
||+---+| | | |||+|/|| | | | |
|+-----+| | ||+---+| | | | |
| | | |+-----+| | | | |
| | | |+-----+| | | | |
+-----+
+- \ ---- / ---- +
+- -:
--| +- \. --- / --- +
+- &. --- |.
+/\ -: +/\.&.|.
```

It is sometimes helpful to define and experiment with parts of a phrase. For example, the tautology m7 asserts the equivalence of the prefix sum scan +/\ and the suffix sum scan +/\. under reversal. Thus:

```
pss=: +/\
x=: 1 2 3 4 5 6
pss x NB. Subtotals, or partial sums
1 3 6 10 15 21

sss=: +/\.
sss x
21 20 18 15 11 6

sss |. x
21 15 10 6 3 1

|. sss |. x
1 3 6 10 15 21

sss&.|. x
1 3 6 10 15 21

<\x NB. Box prefix scan
+-----+
|1|1 2|1 2 3|1 2 3 4|1 2 3 4 5|1 2 3 4 5 6|
+-----+

<\.x NB. Box suffix scan
+-----+
|1 2 3 4 5 6|2 3 4 5 6|3 4 5 6|4 5 6|5 6|6|
+-----+
```

Although an unfamiliar function should first be applied in simple cases, complex cases should be tried as well. For example:

```
s=: sort=: /:~
numb=: 3 1 4 1 [ char=: 'cage'
poem=: 'i sing','of olaf',: 'glad and big'
(, .s numb);(, .s char);poem;(s poem);s"1 poem
+-----+
|1|a|i sing      |glad and big|      giins|
|1|c|of olaf     |i sing        |      affloo|
```

```
|3|e|glad and big|of olaf      | aabddggiln|
|4|g|                  |          |          |
+-----+

```

Certain utilities included in a J system are normally loaded automatically; if not, they can be invoked by entering `0!0 <'profile.ijs' .` For example:

```
names ''
d3    m1    m2    m4    m5    m6    m7    mean n0

```

The response should include all names defined by your experiments.

Presence of these utilities is assumed throughout.

Fixing Definitions

The use of one or more phrases as components in the definition of another phrase may simplify and clarify the latter definition. For example, Parity and Symmetry includes:

```
n16=: m=: 3 1 4,2 0      A 3-by-3 matrix
5,:1 4 1
d17=: ip=: +/ . *        Inner (matrix) product
m18=: L=: m&ip           A linear function

```

However, re-definition of a component phrase may inadvertently alter the definition of the phrase in which it is used. This may be avoided by using the adverb `f.` to fix a definition, perhaps assigning a new name to it. Thus (assuming the foregoing three definitions to have been made):

```
L
(3 3$3 1 4 2 0 5 1 4 1)&ip

L x=: 1 2 5
25 27 14

```

```

      h=: 'L' f.
      h x
25 27 14

      ip=: -/ . *
      L x
21 27 _2

      h x
25 27 14

```

If, as in the present example, the entity being fixed is a verb, the enclosing quotes may be omitted, as in `L f. .`

1B. Locales And The Loading Of Script Files

The phrases for each section are contained in the subdirectory `system\examples\phrases` in the J system. They can be loaded as illustrated below:

```
load 'system\examples\phrases\phrd1.ijs'
bft
1 : 'i. by i. over x./~@i.'

* bft 4
+-----+
| |0 1 2 3|
+-+-----|
|0|0 0 0 0|
|1|0 1 2 3|
|2|0 2 4 6|
|3|0 3 6 9|
+-----+
```

For most use this load into the base locale is satisfactory. However, in any joint use of phrases from different sections it must be remembered that the names `m0`, `m1`, etc., are unique only within each section. Distinct names can be assigned by loading the files into distinct locales, perhaps using the letter that denotes the section followed by the number that denotes the chapter. For example:

```
'a2' load 'system\examples\phrases\phra2.ijs'
The file for Section A of Chapter 2
m67_a2_
l&o.

m67_a2_ 0 1
0 0.841471

m67
|value error
sin_a2_ 0 1
0 0.841471
```

Any names may be used for locales, but the use of majuscules will avoid conflict with those (such as j) used by the utilities. Simpler names may also be assigned at will, as in

```
q=: m67_a2_
```

For a screen display of any group of phrases (complete with comments), use the file read. Thus:

```
y=: 1!:1 <'system\examples\phrases\phral.ijs'
y
n0=: i. 6          NB.List of first six non-negative
integers
m1=: ^&3           NB.Cube
m2=: mean=: +/ % # NB.Arithmetic mean
d3=: $,: NB.x copies of y
m4=: <.@(0.5&+) NB.Round
m5=: =+           NB.Test for real number
m6=: (<0 _1)&C.    NB.Swap leading and final items
m7=: +/\ -: +/\.&|. NB.Prefix sum scan is suffix under
reversal
```

1C. Specimens

Although many aspects of a phrase may be illustrated by applying it to scalar arguments or arrays of successive integers, more thorough explorations require the use of several classes of arrays, including negative, fractional, complex, alphabetic, and boxed. Moreover, the use of only systematic arguments such as `i. 6` and `i. 2 3 4` might suggest that a phrase possesses properties that it does not when applied to a random array such as `? . 2 3 4$10`.

The phrases of this section provide a variety of specimen arrays. Because they will be used in illustrating the application of phrases in many different sections, they should perhaps be loaded into the special `z` locale, so as to be generally available, either in the form `d0_z_` or `d0`

<code>d0=: QI=: ?.@\$</code>	Shape x of integers from <code>i. y</code>
<code>d1=: QD=: ?.@(, :~)@\$</code>	Shape 2 , x of integers from <code>i. y</code>
<code>d2=: QN=: -/@QD</code>	Random negative and positive integers
<code>d3=: QF=: %/@QD</code>	Random fractions
<code>d4=: QC=: j./@QD</code>	Random complex numbers
<code>d5=: QA=: {&a.@(a.&i.@]+[QI 26"_)</code>	Random literal (26 beginning at <code>y.</code>)
<code>m6=: QI&10</code>	Array of random single-digit integers
<code>m7=: QZ=: QI&1</code>	Array of zeros
<code>m8=: QB=: QI&2</code>	Array of Booleans

Random arrays can be produced by expressions of the form `? 2 3 4$10`, but for repeatable experiments it is necessary to use a specific random seed, as provided by the function `? .`, and in any other phrase that uses it. For example:

```
(QA&'A' ; QA&'a' ; QA&'+' ) 2 3 4
+-----+
|DTLN|dtln|. >68|
|FBRR|fbr|. 0,<<|
|YJNV|yjnv|C48@|
|      |      |      |
```

```
|ABNR|abnr|+,8<|
|AJBK|ajbk|+4,5|
|RPYW|rpyw|<:CA|
+-----+
```

The following phrases may be applied to box an argument, to scale and shift it to a specified range, or to normalize it to the range 0 to 1:

a9=: B=: 1 : '<"x.'	Adverb: k B is box of rank k
m10=: B0=: 0 B	Box atoms
m11=: B1=: 1 B	Box vectors (lists)
m12=: B2=: 2 B	Box matrices (tables)
m13=: RG=: (1: + >./ - <./)@,	Range of values of y
d14=: SC=: * % RG@]	Scale range of y to scalar x
m15=: NR=: 1&SC	Normalize range to 1
m16=: SZ=: - <./@,	Shift to make smallest value zero
m17=: NM=: SZ@NR	Normalize to range 1 and min value 0

```
      B0 B1 2 3 4 QA 'A'
+-----+
|+----+|+----+|+----+| | | | | | |
||DTLN|||FBRR|||YJNV||
|+----+|+----+|+----+|
+-----+-----+-----+
|+----+|+----+|+----+| | | | | | |
||ABNR|||AJBK|||RPYW||
|+----+|+----+|+----+|
+-----+
```

```
      B2 3 4 5 QA 'a'
+-----+
|dtlnf|rpywn|zstqb|
|brryj|crksx|qxhlt|
|nvabn|tgbti|mghje|
|rajbk|qtzjg|mxxbx|
+-----+
```

```
(] ; NR ; SZ ; NM) 3 4 QN 10
```

```

+-----+
|_4  2 _1 0|1.6 0.4 0.1    0|1 7 4 5|0.1 0.7 0.4 0.5|
|_3 _5  1 1|0.9 2.5 0.1 0.1|2 0 6 6|0.2    0 0.6 0.6|
| 4 _2  0 3|1.6 0.4    0 0.9|9 3 5 8|0.9 0.3 0.5 0.8|
+-----+
] Scale range to 1 Shift to 0 min Norm pos and range

```

1D. Function Tables

d0=: +/	Addition table
d1=: */	Multiplication table
d2=: >./	Maximum table
d3=: [by] over +/	Bordered addition table
d4=: by=: ' '&@, .@[,.]	Format function
d5=: over=: ({ . ; } .) @ " : @ ,	Format function
m6=: +/~@i.	Addition table on first y integers
m7=: bc=: !/~@i.	Binomial coefficients of order y
a8=: ft=: 1 : 'x./~@i.'	Function table
a9=: bft=: 1 : 'i. by i. over x./~@i.'	Bordered function table (Explicit definition)

The first three phrases produce function tables as illustrated below:

```

      2 3 5 (d0 ; d1 ; d2) 0 1 2 3 4 5
+-----+
| 2 3 4 5 6 7 | 0 2 4 6 8 10 | 2 2 2 3 4 5 |
| 3 4 5 6 7 8 | 0 3 6 9 12 15 | 3 3 3 3 4 5 |
| 5 6 7 8 9 10 | 0 5 10 15 20 25 | 5 5 5 5 5 5 |
+-----+

```

The next produces an addition table bordered by its arguments for easy reading; m6 and m7 produce tables on lists of integers, and the adverb a9 produces a bordered function table:

```

      2 3 5 d3 0 1 2 3 4 5
+-----+
| | 0 1 2 3 4 5 |
+--+-----+
| 2 | 2 3 4 5 6 7 |
| 3 | 3 4 5 6 7 8 |
| 5 | 5 6 7 8 9 10 |

```

+-----+

(m6 ; m7) 5

+-----+

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

+-----+

% a9 5

+-----+

	0	1	2	3	4	
+ +	+-----+					
0	0	0	0	0	0	
1	_	1	0.5	0.33333333	0.25	
2	_	2	1	0.66666667	0.5	
3	_	3	1.5	1	0.75	
4	_	4	2	1.33333	1	

+-----+

2A. Bond or Curry

Fixing an argument of a verb produces a monad. Some verbs so produced are sufficiently important to justify being denoted by a primitive symbol, and the following table often shows the corresponding primitive together with the English definition. The conjunction $\&$ (often called *with*) is used to bond an argument to a verb.

m0 =: 1&+	Increment >:
m1 =: +&1	"
m2 =: _1&+	Decrement <:
m3 =: -&1	"
m4 =: 1&-	Not -. (logical and probability complement)
m5 =: 1&~:	"
m6 =: 0&=	"
m7 =: 0&-	Negate - (arithmetic)
m8 =: _1&*	"
m9 =: *&_1	"
m10=: 2&*	Double +:
m11=: *&2	"
m12=: 3&*	Triple
m13=: *&3	"
m14=: 0j1&*	j. (Multiply by $\sqrt{-1}$)
m15=: ^@j.	r. (Complex # on unit circle at y radians)
m16=: 1p1&*	π times
m17=: 0.5&*	Halve -:
m18=: *&0.5	"
m19=: %&2	"
m20=: 1&%	Reciprocal %
m21=: ^&_1	"

m22=: ^&2	Square *:
m23=: ^&33	Cube
m24=: ^&0.5	Square root %:
m25=: ^&1r2	"
m26=: 2&%:	"
m27=: ^&(%3)	Cube root
m28=: ^&1r3	"
m29=: 3&%:	"
m30=: (^1)&^	Exponential ^
m31=: 1x1&^	"
m32=: 1x1&^.	Natural log ^.
m33=: 10&^	Antilog
m34=: 10&^.	Base-10 log
m35=: >:@<.@(10&^.)@(1&>.)	# of digits needed to represent integer y
m36=: #@ (10&#. ^:_1) "0	"
m37=: >:@<.@(2&^.)@(1&>.)	# of bits needed to represent integer y
m38=: #@ (2&#. ^:_1) "0	"
m39=: 0&{	Head (first) { .
m40=: _1&{	Tail (last) { :
m41=: 1&}.	Behead } .
m42=: _1&}.	Curtail } :
m43=: 0&<	Positive test
m44=: 0&>	Negative test
m45=: 0&>.	Max (0,y)
m46=: 0&<.	Min (0,y)
m47=: (0&=)@(2&)	Even test
m48=: (1&=)@(2&)	Odd test
m49=: _1&A.	Reverse .
m50=: (<0 _1)&C.	Interchange first and last items
m51=: >.@(0.5&+)	Round

m52=: ,~ \$ 1: ,] \$ 0:	Identity matrix of order y
m53=: -.@(' '&E.) #]	Remove multiple blanks
m54=: BC=: i.@>: !]	Binomial coefficients of order y
m55=: (0&,+,&0)^:([`1:)	" (recursive)
m56=: BCT=:i. !/ i.	BC table of orders to y-1
m57=: PAT=: :@BCT	Pascal's triangle
m58=: (0&,+,&0)^:(i.`1:)	" (recursive)
m59=: IX=: a.&i.	Index in ASCII alphabet
m60=: Lt=(1&e.)@(e.&a.)@,	Literal test
m61=: 1&#.	Sum over lists (last axis) +/"1
m62=: 1& ,	Preface a row of 1's
m63=: ,&1	Append a row of 1's
m64=: 1& ,.	Preface a column of 1's
m65=: ,.&1	Append a column of 1's
m66=: 1&,@\$ \$,	Itemize (append leading 1 to shape) , :
m67=: sin=: 1&o.	Sin
m68=: asin=: _1&o.	Arcsin
m69=: cos=: 2&o.	Cos
m70=: acos=: _2&o.	Arccos
m71=: tan=: 3&o.	Tan
m72=: atan=: _3&o.	Arctan
m73=: sinh=: 5&o.	Sinh
m74=: asinh=: _5&o.	Arcsinh
m75=: cosh=: 6&o.	Cosh
m76=: acosh=: _6&o.	Arccosh
m77=: tanh=: 7&o.	Tanh
m78=: atanh=: _7&o.	Arctanh

2B. Rank

m0=: +/	sum (column sums of a matrix)
m1=: +/"1	sum vectors (row sums of a matrix)
d2=: *"0 _	outer product; same as */
m3=: +/ @ (*"1 _)	inner product; same as +/ .*

2C. Ambivalent Functions

The phrase $h =: f : g$ defines h as the function whose monadic case is f and whose dyadic case is g . The components f and g may be functions already defined and named, or they may be tacit or explicit phrases. Moreover, either (but not both) may be defined in terms of the other by using $\$$ for self-reference in a tacit definition.

<code>v0=: 10&^. : ^.</code>	Base 10 log for monadic case
<code>v1=: 10&\$: : ^.</code>	Same using self-reference to dyad
<code>v2=: 10&^. : (\$:*^.@(10"0)%^.@[)</code>	Same using self-reference to monad
<code>d3=: res=: [: : </code>	Domain of monad is empty (dyadic only)
<code>m4=: abs=: : [:</code>	Domain of dyad is empty (monadic only)

2D. Adverbs and Conjunctions

Adverbs from Conjunctions

A conjunction together with one of its arguments produces an adverb defined in an obvious way. For example, if $a1 =: \&3$, then $\wedge a1$ is equivalent to $\wedge \&3$ (the cube function), and if $a2 =: 3\&$ then $\wedge a2$ is equivalent to $3\&^$ (the three-to-the-power function). It is therefore easy to define useful families of adverbs from a conjunction, so easy that it is fruitless to attempt an exhaustive catalogue. The following list is intended to suggest the possibilities in various classes:

$a0 =: I =: \wedge :_1$	Inverse ($\wedge I$ is \wedge .)
$a1 =: L =: \wedge :_$	Limit ($2\&o.L$ 1 for soln of $y=\cos y$)
$a2 =: LI =: \wedge :_$	Limit of inverse
$a3 =: SQ =: \wedge :^2$	Square ($1\&o.SQ$ for sine squared)
$a4 =: C =: \&o.$	Family of circular fns ($3 C$ is tangent)
$a5 =: CO =: \%@C$	$3 CO$ is cotangent
$m6 =: rfd =: 1r180p1\&*$	Radians from degrees
$m7 =: dfr =: rfd I$	Use $dfr =: dfr f.$ to fix definition
$a8 =: D =: @rfd$	Try $1 C D 0 30 45 60 90 180$
$m9 =: SIN =: 1\&o. D$	Sine for degree arguments
$a10 =: T =: "2$	Try $<T I. 2 3 4 3$ (BOX TABLES)
$a11 =: S =: \wedge !.$	Stope (rising or falling factorial fn etc)
$a12 =: P =: p !.$	Stope polynomial
$a13 =: FILL =: !.$	Fill for shift (non-cyclic rotate)
$a14 =: FILE =: 1 !:$	File functions ($1 FILE$ for read, etc.)

Explicit Definitions

An adverb or a conjunction can be defined in explicit form. For example:

```

split=: 2 : ' ,.@(x.@(y.&{.) ; x.@(y.&{.})) '
]x=: i. 5 3
0 1 2
3 4 5
6 7 8
9 10 11
12 13 14

```

```

(+: split 2 ,. |. split 3 ,. +/ split 2) x
+-----+
|0 2 4 |6 7 8 |3 5 7 |
|6 8 10 |3 4 5 |
| |0 1 2 |
+-----+-----+
|12 14 16|12 13 14|27 30 33|
|18 20 22| 9 10 11|
|24 26 28|
+-----+

```

c15=: split=: 2 : ' ,.@(x.@(y.&{.) ; x.@(y.&{.})) '	split as defined above
d16=: by=: ' '&@, .@[,.]	Verb for use in the table adverb below
d17=: over=: ({ . ; } .) @ " : @ ,	Verb for use in the table adverb below
a18=: table=: 1 : '[by]over x. / '	Try 1 2 3 * table 4 5 6 7

Noun Arguments

Adverbs that apply to a noun argument, and conjunctions that apply to one noun argument and one verb argument are commonplace. For example:

```

x=: 0 0 1 1 [ y=: 0 1 0 1
x *. y Boolean and
0 0 0 1

```

```

x 1 b. y Boolean adverb

```

```
0 0 0 1
```

```
    x (i.16) b. y All sixteen Boolean functions
```

```
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

```
    C=: 1 : 'x.&o.' Circle adverb
    1 C 0 1r4p1 1r3p1 1r2p1 1p1 Sine function
0 0.7071068 0.8660254 1 0
```

```
    ^&3 x=: i. 6 Cube
0 1 8 27 64 125
```

```
    2 |. !. 1 x Shift in ones
2 3 4 5 1 1
```

Conjunctions that apply to two nouns are less familiar, although the definitions of functions in terms of nouns occur frequently in math. For example, a *rational* function (the quotient of a polynomial $a \&p.$ divided by another $b \&p.$) is defined by a pair of coefficients. Thus:

```
    a=: 1 4 6 4 1 [ b=: 1 2 1
    RAT=: 2 : 'x.&p. % y.&p.'
    a RAT b
1 4 6 4 1&p. % 1 2 1&p.
```

```
    a RAT b y=: i.6
1 4 9 16 25 36
```

```
    b RAT a y
1 0.25 0.111111 0.0625 0.04 0.0277778
```

```
    (a RAT b * b RAT a) y
1 1 1 1 1 1
```

We may also remark that expressions such as $2x^3$ and $2x^3 + 4x^2$ are commonly used in elementary math to define functions rather than to indicate explicit computation: the x in the foregoing can be construed (and defined) as a conjunction such that $2x3$ is the function $2: *]^3:$. Thus:

```

x=: 2 : 'x.&* @ (^&y.) " 0'
2 x 3
2&*@(^&3)"0

```

```

2 x 3 y=: 0 1 2 3 4 5
0 2 16 54 128 250

```

```

2 * y ^ 3
0 2 16 54 128 250

```

```

2 3 5 x 1 2 4 y
0 0 0
2 3 5
4 12 80
6 27 405
8 48 1280
10 75 3125

```

The last result above gave the values of the individual terms; in order to obtain their sums (and yet retain the behaviour for a single term), we redefine the conjunction x as follows:

```

x=: 2 : '+/ @ (x.&* @ (^&y.)) " 0'
2 3 5 x 1 2 4 y
0 10 96 438 1336 3210

```

```

(2 x 1 + 3 x 2 + 5 x 4) y
0 10 96 438 1336 3210

```

```

2 x 3 y
0 2 16 54 128 250

```

c19=: RAT=: 2 : 'x.&p. % y.&p.'	Produces rational function
c20=: x=: 2 : '+/ @ (x.&* @ (^&y.)) " 0'	Mimics notation of elementary math
c21=: bind=: 2 : 'x. @ (y."_)'	Binds y to the monad x

It is often convenient to bind an argument to a monad, producing a function that ignores its argument. For example, using `wdinfo` , a monad that displays its argument in a message box, the definition `fini=: wdinfo bind 'Job Finished'` produces a function such that `fini''` is equivalent to `wdinfo 'Job Finished'` .

2E. Explicit Definitions

Explicit definition is convenient for anyone more familiar with conventional programming, particularly that using if-then-else forms. It is also convenient for anyone engaged in extending their mastery of tacit programming: the automatic translation of a one-line explicit definition to tacit form can provide instruction in the reading and writing of tacit expressions. For example:

```
log=: 3 : '10 ^. y.'
log 1 10 20 40 100
0 1 1.30103 1.60206 2
```

```
log Display of the definition of log
3 : '10 ^. y.'
```

```
log=: 13 : '10 ^. y.'
log 1 10 20 40 100
0 1 1.30103 1.60206 2
log Display of the (tacit) definition of log
10"_ ^. ]
```

a0=: def=: : 0	Adverb for entering explicit definitions
----------------	--

The phrases 1 : 0 and 2 : 0 and 3 : 0 may be used for entering explicit definitions of adverbs, conjunctions, and functions, without entering enclosing quotes. The adverb `def` defined above makes their use somewhat more convenient. For example:

```
LOG=: 3 def
10 ^. y.
:
x. ^. y.
)
```

```
LOG 10
1
2 LOG 32
5
```

```

    rat=: 2 def
x.&p. % y.&p.
)

    1 4 6 4 1 rat 1 2 1
1 4 6 4 1&p. % 1 2 1&p.

    (1 4 6 4 1 rat 1 2 1) i. 6
1 4 9 16 25 36

```

It is important to recognize that explicit definitions are ordinary J statements (using the conjunction :) , and can be used with other expressions. For example:

```

    mat=: [;._2 (0 : 0) Define matrix
one
two
three
)

    boxed=: <;._2 (0 : 0) Define boxed list
one
two
three
)

    fn=: 3 : 0"1 Assign rank to an explicitly defined function
< +/ y.
)

    x=: 3
    3 : 0 x Execute unnamed explicit definition
if. 2|y. do. 'odd' else. 'even' end.
)
odd

    mat
one
two
three

    $mat
3 5

```

boxed

```
+-----+  
|one|two|three|  
+-----+
```

fn i. 3 4

```
+-----+  
|6|22|38|  
+-----+
```

2F. Hooks

A pair of functions in isolation form a hook whose monadic case is defined as in:

$(= <.)$ y is equivalent to $y = <. y$

hence the function $h=: =<.$ compares its argument with its integer part, and therefore provides a test for integers. Hooks occur frequently in most sections.

m0=: It=: =<.	Integer test
m1=: Rt=: =+	Real test
d2=: \$,:	x copies of y
d3=: \$,	Reshape as in APL
m4=: cf=: (+%)/	Continued fraction
m5=: cfc=: (+%)/\	Continued fraction convergents
m6=: ifb=: # i.@#	Integers from boolean list
m7=: [m	([m)x invokes m y, then returns y

For example:

```
cf 3 7 15 1 Approximation to pi
3.14159
```

```
cfc 3 7 15 1 Convergents to pi
3 3.14286 3.14151 3.14159
```

```
cfc 1 1 1 1 1 1 1 Convergents to golden mean
1 2 1.5 1.66667 1.6 1.625 1.61538
```

```
cfc 10$1x As above in extended precision
1 2 3r2 5r3 8r5 13r8 21r13 34r21 55r34 89r55
```

2G. Verb Trains

The following phrases illustrate how trains may be substituted for other constructs, often giving expressions that are more readable, particularly to readers more familiar with expressions from mathematics:

m0=: >: @ +: @ i.	First odd integers
m1=: 1: + 2: * i.	Same as m0
m2=: +/ @ (1: + 2: * i.)	Sum of odd integers
m3=: [: +/ 1: + 2: * i.	Same as m2 using cap
m4=: m2 -: *:	Sum of odds is square (Tautology)
m5=: < i. 1:	The first element where $x < y$; also works for $=$ \sim : < <: >: > E.
m6=: < i. 0:	The first element where not $x < y$
m7=: < i: 1:	The last element where $x < y$
m8=: < i: 0:	The last element where not $x < y$
m9=: [: +/ <	The number of elements where $x < y$

2H. Gerunds

The phrases included here illustrate the following uses of gerunds:

- Insertion
- Case statement
- Recursion
- Power

m0=: horner=: +`*/	m0 a,x,b,x,c is (a,b,c)p.x
m1=: grid=: +`(*i.)/	grid b,s,n From b in n s-steps
m2=: casel=: _1:`%:`*:@.*"0	Sqr, _1, or sq root if neg, zero, or pos
d3=: sort=: /:~@]`(\:~@])@. [Sort up or down for left 0 or 1
v4=: cases=: casel : sort	Ambivalent function
a5=: sel=: 1 : '] #~] x. {.'	Selection for Quicksort
m6=: qs=:] ` (\$:@(<sel), =sel, \$:@(>sel)) @. (1:<#)	Quicksort defined recursively
m7=: (0&,+,&0)^:([`1:)	Binomial coefficients (gerundial power)

3A. Indexing

{ encompasses everything performed by bracket indexing in other languages. The ranks are 0 _ , that is, $x\{y$ can be considered individually for each scalar in x and for y *in toto*, with the overall result constructed from the individual results in the same way as for all other functions. For scalar x then:

$>x$ is a scalar or a list, and $r=:>j\{ ,>x$ are the indices for axis j . r may be integers in the range $i.\&.(+&n)n=:j\{\$y$, which selects cells $n|r$, or r may be boxed integers in that range, in which case the selected cells are all except those in $n|r$. Thus:

```

z=: 0{y=: 3 3 3$'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
ibbb=: <ibb=: <ib=: <i=: 1 _1
jbbb=: <jbb=: <jb=: <j=: 2 1
ijbbb=: <ijbb=: <ijb=: <ij=: 2 2$ i,j
(] ; i&{ ; ib&{ ; ibb&{ ; ibbb&{) z
+-----+
|ABC|DEF|F|DEF|ABC|
|DEF|GHI| |GHI|  |
|GHI|  | |  |  |
+-----+

    ijb{y
|rank error
|    ijb    {y

    (] ; i&{ ; ijbb&{ ; ijbbb&{) z
+-----+
|ABC|DEF|DEF|ABC|
|DEF|GHI|GHI|  |
|GHI|  |  |  |
|  |  |GHI|  |
|  |  |DEF|  |
+-----+

```

The *amend* adverb } applied to an index produces a function that replaces the selected part of the right argument by the left argument. For example:

```

    '*' ib} z
ABC
DE*
GHI

```

```

    ('def',: 'ghi') i} z
ABC
def
ghi

```

```

    (] ; i&{ ; ib&{ ; ibb&{ ; ibbb&{)"2 y
+-----+
|ABC|DEF|F|DEF|ABC|
|DEF|GHI| |GHI| |
|GHI| | | | |
+---+---+---+---+
|JKL|MNO|O|MNO|JKL|
|MNO|PQR| |PQR| |
|PQR| | | | |
+---+---+---+---+
|STU|VWX|X|VWX|STU|
|VWX|YZ]| |YZ]| |
|YZ]| | | | |
+-----+

```

```

    $(<<'') { z
0 3

```

```

    $(<a:) { z
0 3

```

Indexing on higher-rank arrays may be illustrated by the argument `y`:

```

    ]k=: <1 2;a:;0 2
+-----+
|+-----+|
||1 2|++|0 2|| | |
|| | ||| ||
|| |++| ||
|+-----+|
+-----+

```

```

y ; k{y

```

```

+-----+
|ABC|JL|
|DEF|MO|
|GHI|PR|
|   |   |
|JKL|SU|
|MNO|VX|
|PQR|Y]|
|   |   |
|STU|   |
|VWX|   |
|YZ]|   |
+-----+

```

The following examples further illustrate the use of the indexing function. For each example, it may be instructive to plug the values into the expression $r = :>j\{ ,>x$ and work out the result.

n0=: y=: i.4 5 6 7	Array used in examples
n1=: (<,<3){y	Item 3 of y
n2=: (<,3){y	Item 3 of y
n3=: (<3){y	Item 3 of y
n4=: 3{y	Item 3 of y
n5=: (<,<_1){y	The last item of y (item _1 of y)
n6=: (<,_1){y	The last item of y (shape 5 6 7)
n7=: (<_1){y	The last item of y
n8=: _1{y	The last item of y
n9=: (_1+#y){y	The last item of y
n10=: 0{y	The first item of y
n11=: (-#y){y	The first item of y
n12=: 3 0 _2 0{y	Items 3 0 _2 0 of y
n13=: i=: ?2 3\$0{\$y	Indices used in examples
n14=: j=: ? 1{\$y	Indices used in examples
n15=: k=: ?7 \$2{\$y	Indices used in examples
n16=: (<i;j;k){y	y[i;j;k] in APL notation
n17=: (<1;2;3){y	y[1;2;3;]
n18=: (<1,2,3){y	y[1;2;3;]

n19=: (<1 2 3){y	y[1;2;3;]
n20=: (<<i){y	y[i;;;]
n21=: (<<,i){y	y[,i;;;]
n22=: (,i){y	y[,i;;;]
n23=: (<<1 3 2){y	Items 1 3 2
n24=: (<<<1 3 2){y	All but items 1 3 2
n25=: (<<<1 3){y	All but items 1 3
n26=: (<<<1){y	All but items 1
n27=: (<<<\$0){y	All but items ... none; i.e. all items
n28=: (<<a:){y	All items
n29=: (<1 3 2;3){y	y[1 3 2;3;;;]in APL (0-origin)
n30=: (<(<1 3 2);3){y	y[(i.#y)-.1 3 2;3;;;]
n31=: (<(<1 3);3){y	y[(i.#y)-.1 3;3;;;]
n32=: (<(<1);3){y	y[(i.#y)-.1;3;;;]
n33=: (<(<\$0);3){y	y[(i.#y)-.\$0;3;;;]
n34=: (<(<\$0);3){y	y[;3;;;]
n35=: (<a:;3){y	y[;3;;;]
n36=: 4{"_1 y	y[;4;;;]
n37=: (<a:;a:;5){y	y[;;5;;;]
n38=: 5{"_2 y	y[;;5;;;]
n39=: (<1 2){y	Abbreviated (fewer indices than axes)
n40=: _2{y	Negative
n41=: (<<<3){y	Complementary
n42=: (1 2;3 2;0 _2){y	Scattered (non-scalar left argument)

3B. Merge and Amend

Two arguments x and y can be merged by interleaving their items (necessarily of a common shape) as determined by a Boolean list of shape x +– y . For example:

```
x=: >: 'That they hunted from hill'
y=: >: 'second time me to plain'
b=: 0 1 1 0 0 1 0 0 1 1
mrg=: 1 : ' /: @ /: @ (x. "_ ) { , '
x([ ; ] ; ( ,. b) "_ ; b mrg) y
```

```
+-----+
|That  |second|0|That  |
|they  |time  |1|second|
|hunted|me    |1|time  |
|from  |to    |0|they  |
|hill  |plain |0|hunted|
|      |      |1|me    |
|      |      |0|from  |
|      |      |0|hill  |
|      |      |1|to    |
|      |      |1|plain |
+-----+
```

The form of the function `b mrg` obtained by applying the adverb `mrg` suggests the form of a function `MRG` to be applied to a Boolean left argument and a right argument formed as the catenation of the original arguments. Thus:

```
b mrg
/: @ /: @ (0 1 1 0 0 1 0 0 1 1 "_ ) { ,
```

```
MRG=: /: @ /: @ [ { ]
b MRG x, y
```

```
That
second
time
they
hunted
me
from
```

```
hill
to
plain
```

The argument `b` need not be Boolean, but may be anything of the requisite number of items that is in the domain of `/:`. For example:

```
b=: 0 2 2 1 0 2 2 2 0 0 1 1 2 1 2 1 1 1 1
y0=: 'abcd' [ y1=: '123456789' [ y2=: 'zzzzzzzz'
b MRG y0,y1,y2
azz1lbzzzcd23z4z56789
```

a0=: mrg=: 1 : '/:@/:@(x."_) { , '	x b mrg y merges x and y
m1=: MRG=: /:@/:@[{]	b MRG x,y is equivalent to above
d2=: alt=: ,@,.	Merge items from x and y alternately

For example:

```
x=: 'temr rtes'
y=: 'h axbohr '
x alt y
the marx brothers
```

An argument can be amended by replacing those cells selected by an index, by the cells of another argument. For example:

```
x=: 'ABCD' [ y=: 'abcdefghij'
i=: 4 2 8 6
i{y
ecig

]z=: x i} y
abBdAfdhCj

m=: a.{~(a. i. 'A')+i.5 5
]i=: 2 # &.> i. # m
+-----+
```

```

|0 0|1 1|2 2|3 3|4 4|
+-----+

x=: '+-.*%^'
m ; (i{m) ; x ; x i} m
+-----+
|ABCDE|AGMSY|+-.**%^|+BCDE|
|FGHIJ|      |      |F-HIJ|
|KLMNO|      |      |KL*NO|
|PQRST|      |      |PQR%T|
|UVWXY|      |      |UVWX^|
+-----+

```

Amendment can also be made by using a function that selects a portion of its argument. For example:

```

IR=: @(i.@$@]) Adverb to select indices of (ravelled) table
right argument
A=: IR } Adverb to amend selected portion of right argument
d=: (<0 1)&|: Function to select diagonal of a table
'+-.*%^' (] ; d@] ; ]IR ; d IR ; d IR } ; d A) m
+-----+
|ABCDE|AGMSY| 0  1  2  3  4|0 6 12 18 24|+BCDE|+BCDE|
|FGHIJ|      | 5  6  7  8  9|      |F-HIJ|F-HIJ|
|KLMNO|      |10 11 12 13 14|      |KL*NO|KL*NO|
|PQRST|      |15 16 17 18 19|      |PQR%T|PQR%T|
|UVWXY|      |20 21 22 23 24|      |UVWX^|UVWX^|
+-----+
m      diag m      Indices of ravelled m      Indices of
diagonal Amendments

```

```

ur=: 2 _3&{. Select upper right corner
(2 3$'+-.*%^!') (] ; ur@] ; ]IR ; ur IR ; ur IR } ; ur A) m
+-----+
|ABCDE|CDE| 0  1  2  3  4|2 3 4|AB+-*|AB+-*|
|FGHIJ|HIJ| 5  6  7  8  9|7 8 9|FG%^!|FG%^!|
|KLMNO|    |10 11 12 13 14|    |KLMNO|KLMNO|
|PQRST|    |15 16 17 18 19|    |PQRST|PQRST|
|UVWXY|    |20 21 22 23 24|    |UVWXY|UVWXY|
+-----+

```

a3=: IR=: @(i.@\$@])	f IR selects indices of ravelled rgt arg
----------------------	--

m4=: d=: (<0 1)& :	Function to select diagonal of table
m5=: ur=: 2 _3&{.	Function to select upper right corner

4A. Intervals

It is a common need in programming to have to test whether a given number x lies between two other numbers a and b , with a less than b , called the lower boundary and upper boundary. The numbers that lie between the boundaries are called an interval. Each boundary may or may not be included in the interval. If the boundary is included in the interval, the interval is said to be closed on that side; if it is excluded it is said to be open on that side. If an interval is closed on both sides, it is said to be a closed interval. If it is open on both sides it is said to be an open interval. If it is open on the left side, it is said to be half open on the left. If it is open on the right it is said to be half open on the right. This is usually expressed in common mathematical notation in one of the following ways, which permit all possible ways in which the boundary numbers are or are not included in the interval:

set notation	interval notation	description of interval
$\{x \mid a < x < b\}$	(a, b)	open
$\{x \mid a < x \leq b\}$	$(a, b]$	half-open on the left
$\{x \mid a \leq x < b\}$	$[a, b)$	half-open on the right
$\{x \mid a \leq x \leq b\}$	$[a, b]$	closed

In the usual case, a and b are finite numbers, with a less than b . If we permit the boundaries to be infinite, and a to be equal to b , we get the cases tabulated below. See Andrew M. Gleason, *Fundamentals of Abstract Analysis*, Addison-Wesley, 1966, sect. 14-10.7

set notation	interval notation	description of interval
$\{x \mid a < x\}$	$(a, _)$	x greater than a
$\{x \mid x < b\}$	$(_, b)$	x less than b
R	$(_, _)$	x all finite real numbers
$\{x \mid x \leq b\}$	$(_, b]$	x less than or equal to b
$\{x \mid a \leq x\}$	$[a, _)$	x greater than or equal to a
$\{a\}$	$[a, a]$	x equal to a
$\{\}$	(a, a)	x is the empty set

In the verbs below the limits a and b form the two items of the right argument y. The verbs d0 through d3 below are reasonably efficient and allow testing for inclusion in an interval by using the relational symbols < and ≤ in pairs in all four possible ways. The verbs d5 through d8 use a common subverb d9 which when applied between arguments x and y yields a result between $-\frac{1}{2}$ and $\frac{1}{2}$, depending on whether x is less than a, equal to a, strictly between a and b, equal to b, or greater than b, thus permitting a wide variety of tests. It subtracts y from x, yielding a two-atom result, and takes the sum of the signum of this. They are not particularly efficient, but are interesting pedagogically. The verbs d10 and d11 do not generalize readily, but may be useful in the special case of a need for a half-open left interval. All of these verbs may be used with Boolean, integer, or real arguments. When used with real arguments, you may wish to consider whether the relations should be fuzzed or made with zero tolerance.

d0=: OO=: ({.@] < [)*.([< {:@])	x OinO y (Is x in open interval y)
d1=: OC=: ({.@] < [)*.([<: {:@])	x OinC y
d2=: CO=: ({.@] <: [)*.([< {:@])	x CinO y
d3=: CC=: ({.@] <: [)*.([<: {:@])	x CinC y
d4=: +/"1@d2	Number of x OinC y
d5=: 0: e.~ class	x OinO y
d6=: 0 1"_ e.~ class	x OinC y
d7=: _1 0 "_ e.~ class	x CinO y
d8=: _1 0 1"_ e.~ class	x CinC y
d9=: class=: [: +/"1 [: * -/	Class $-\frac{1}{2}$ to $\frac{1}{2}$: <A ="a" CINC="b">b
d10=: [: ~:/ >:/~	x OinC y
d11=:] >/ .>~ [: ,.~ [x OinC y
d12=: sgd=: *@(-~/~)	Signum of difference
d13=: 0&=@(+/@sgd	OO
d14=: e.&0 1@(+/@sgd	OC
d15=: 0&>:@(%/@sgd	CO
d16=: >/@sgd	CC

```

      x=:i.12 [ y=: 3 8
      x ([ , OO , OC ,CO ,: CC) y
0 1 2 3 4 5 6 7 8 9 10 11
0 0 0 0 1 1 1 1 0 0 0 0 ( )
0 0 0 0 1 1 1 1 1 0 0 0 ( ]
0 0 0 1 1 1 1 1 0 0 0 0 [ )
0 0 0 1 1 1 1 1 1 0 0 0 [ ]

```

```

      x(CC = OC +. CO) y Tautology
1 1 1 1 1 1 1 1 1 1 1 1

```

```

      x (d5 = OO) y
1 1 1 1 1 1 1 1 1 1 1 1

```

```

      x ,: x d9 y
0 1 2 3 4 5 6 7 8 9 10 11
_2 _2 _2 _1 0 0 0 0 1 2 2 2

```

```

      x d12 y
_1 _1 _1 0 1 1 1 1 1 1 1 1
_1 _1 _1 _1 _1 _1 _1 _1 0 1 1 1

```

4B. Locating And Selecting

In dealing with large amounts of data it is frequently necessary to find the location or locations of a particular datum, sometimes a particular value, sometimes one satisfying certain criteria. The verbs below give examples of techniques that can be used. In the first set, indices are the result. The result of each of these can be used to select the item indexed, using an expression such as $(f\ y)\{y$ or $(x\ f\ y)\{y$.

m1=: {.@\:	Index of (IO) first occurrence of maximum of y
m2=: i.>./	IO first occurrence of maximum of y
m3=: {:@/:	IO last occurrence of maximum of y
m4=: {.@/:	IO first occurrence of minimum of y
m5=: {:@\:	IO last occurrence of minimum of y
m6=: i.<./	IO first occurrence of minimum of y
d7=: ~: i. 1:	IO first item where x and y differ
m8=: # - 1: #.~] = ' ' "_	IO last nonblank in y
d9=: <./@i.	IO first occurrence in x of any item of y
d10=: [+ i.&1@}.	IO first 1 in y after first x items
m11=:] i. 1:	IO first 1 in y
d13=: <:@(\$@)) - .@] i. [IO last item of x in y
d14=: (<:@(#@[) - .@[i.])"1	IO last occurrence of y in x
d15=: .@[i.]	IO last occurrence of y in x, from rear
d16=: E. # i.@#@]	Indices of (ISO) beginnings of x in y
d17=: e.~&, # i.@#@]	ISO all occurrences of items of y in x
d18=: =#[:i. [:#[ISO all occurrences of items of y in x
m19=: +/{.\:	ISO 1s in Boolean list y
m20=: [: m38 m39	ISO 1s in Boolean table y
m21=:] i."1 ' ' "_	ISO first blank in rows of table y
m22=: ([:(</\&. .i.1:)' ' "_ ~:])"1	ISO last nonblank in rows of table y
m23=: <:@(+/\) i. i.@(+/)	ISO 1s in Boolean list y
m24=:] # [: i. #	ISO 1s in Boolean list y

m47=: e.&' 0123456789'	Locate digits and blanks in y
d48=: E.	Locate beginning points of pattern x in y
m49=: 2&=@(+/@(0&=@(/~@i.)))	Locate primes less than y
d50=: i.@(#@)e.[Locate indices x in y
d51=:] e.~ [: i. [: # [Locate indices y in x
m52=:] e.~ [: i. [: >: >./	Locate indices y
m53=: */. =&' '	Locate blank rows in table y
m54=:] -: "1 [: { . ' ' " _ ,]	Locate blank rows in table y
m55=: ~:	Locate first instance of each item
m56=: [: */.\ ' ' " _ =]	Locate leading blanks
m57=: [: */.\ . ' ' " _ =]	Locate trailing blanks
m58=: 2: +. /\ 0: , 2: + /\ @ (= & ' ' ' ')	Locate text between and including quotes
m59=: 2: *. /\ 0: , 2: + /\ @ (= & ' ' ' ')	Locate text between quotes
m60=: 2: < /\ 0: ,]	Locate first 1 in each group of 1s in y
m61=: 2: > /\ 0: ,]	Locate first 0 after each group of 1s in y
d62=: * \$ -@] { . 1:	Locate ends of x fields of length y
d63=: * \$] { . 1:	Locate starts of x fields of length y
m64=: 1: , ~ 2: ~: /\]	Loc ends of fields of identical atoms in y
m65=: 1: , 2: ~: /\]	Loc starts of fields of identical atoms in y
m66=: 1: , ~ 2: ~: /\]	Loc where atom differs from next right in y
m67=: 1: , 2: ~: /\]	Loc where atom differs next left in y
m68=:] > [: } . 0: , ~]	Loc last 1 in each group of 1s in y
m69=:] > [: } : 0: ,]	Loc first 1 in each group of 1s
d70=: 0: = [[: i. [: #]	Loc every xth item of y
m71=: 2: [: >: [: i. #	Loc every 2d item of y
d72=: 0: =	Loc atoms of y divisible by x
d73=: [: -. [e.~ [: i.]	Loc atoms of i. y not in x
d74=: [e.~ [: i.]	Loc atoms of i.y in x

d75=: i. = [: # [Loc items of y not in x
m76=: 0: *.]	Loc nothing
d77=: <:	Loc where x implies y
d78=: >	Loc where x but not y
d79=: ~:	Loc exclusive or of x and y
d80=: =	Loc Kronecker delta of x and y
d81=: #@[> /:@(] ,~ i.@#@[)	Loc fills formed by expanding y by x
m82=: (= <./"1) *. (= "1 >./)	Loc saddle points of table y
m83=: [: #: [: i. 2: ^ #	Loc all subsets of order #n (truth table)
m84=: [: #: [: i. 2: ^]	Loc all subsets of order n (truth table)
m85=: 1: [`(m38@)) `(m93@)) }]	Loc indices y in large enough table (connection matrix from table of indices) (inverse of m39)
m86=: 1: [`(<"1@)) `(>:@(>./@))\$0:)@]) }]	Loc indices y in large enough table
m87=: ' '&=	Loc blanks in y
m88=: =	Loc each item of set of y (position or distribution matrix)
m89=: (#~ (1: ~: +/"1))@=	Table in which each row Locs an infix in y of 2 or more consecutive equal items
d90=: [: , [{.&>] (<@"0) 1:	Expansion mask for fields of length y to uniform field of length x
m91=: _8&d90	Expansion mask for fields of length y to uniform right justified fields of length 8
m92=: [: -. [: *./\ . m53	Loc rows preceding trailing blank rows
m93=: 0: \$~ [: >: >./	Loc nothing in table large enough for y
d94=: [-: "1 ([: # [) { ."1]	Loc rows of y beginning with x
m95=: ' '&(+./ .~:)	Loc nonblank columns
m96=: [: -. [: *./\ . ' '"_ =]	Exclude trailing blanks
m97=: m105@m95	Exclude leading blank columns

m98=: +. 1&,@}:	Loc 1s and 1st 0 in each group of 0s
m99=: m98@m95	Exclude all but first blank columns
m100=: *./ .=&' '	Loc blank rows
m101=: 1: , 2: +./\]	Exclude all 0s but the first in a group
m102=: m101@m100	Exclude all but first blank rows
m103=: +./\.	Loc items left of leftmost trailing 0
m104=: m103@m95	Exclude trailing blank columns
m105=: +./\	Loc items right of rightmost leading 0
m106=: *./\	Exclude all 1s right of first 0
m107=: *./\.	Exclude all 1s left of last 0
m108=: +./ .~:&' '	Loc nonblank rows
m109=: m103@m108	Exclude trailing blank rows
m110=: m105@m108	Exclude leading blank rows
m111=: m98@m95	Include only first of group of blank cols
m112=: m98@m108	Include only first of group of blank rows
d113=: +./ .~:"1	Loc rows of x having atoms not y
d114=: -:"1	Loc rows of x matching y
d115=: ,@,. # \$&1 0@#@,	Alternate i{x 1s and i{y 0s
m116=: # \$&1 0@#	Alternate i{x 1s, (1+i){x 0s, etc
m117=: +./\ *. +./\.	Loc atoms between leading & trailing 0s
m118=: *. *./\@(= +./\)"1	Loc first infix of 1s
m119=: ' '&~:	Loc nonblanks
d120=:]*.*.(]e.#)+/\@ (2:</\0:,])@]	Loc groups of ones in y indicated by x
m121=: +.~:/\	Loc 1s in y and 0s between pairs of 1s
m122=: -. *. ~:/\	Loc 0s between pairs of 1s
m123=: </\	Loc leftmost 1
m124=: <:/\	Loc 1s and all 0s following leftmost 0

m125=: ~:/\	Loc where odd number of 1s are at left
-------------	--

If L is a function that determines locations (that is, indices) in an argument according to some criteria, then the function $L \{ \}$ selects the indicated portions of the argument. For example:

d126=: # "1	1s in x select from each row (list) of y
d127=: #.@[{]	Boolean list x as integer selecting from y
d128=: <"1@[{]	Scattered index table x select from y
d129=: {	Selecting x from y
d130=: >./ . #	Select maximum of x located by y
m131=: m41 {]	Select saddle point(s) of y
m132=: ([:- .(1:,' '["_=])#.1:)}].]	Delete trailing blanks
m133=: m104 # "1]	Delete trailing blank columns
m134=: m97 # "1]	Delete leading blank columns
m135=: m110 #]	Delete leading blank rows
m136=: m109 #]	Delete trailing blank rows
m137=: m98@m108 #]	Delete repeated blank rows
m138=: m98@m95 # "1]	Delete repeated blank columns
m139=:] -. [: { . ' '["_ ,]	Delete blank rows
m140=: m108 #]	Delete blank rows
m141=: m95 # "1]	Delete blank columns
m142=: ([: m117] ~: ' '["_) #]	Delete leading & trailing blanks
m143=: +/ @ (* . / \ @ (' ' & =)) } .]	Delete leading blanks
m144=: #~ ([: + . / \ . ' ' & ~:)	Delete trailing blanks
d145=: ,	Append atom or list to table
d146=: , .	List or atom as new column of table
m147=: ~.	Delete repeated items
m148=: */@}:@\$	Number of rows in array y

m149=: #~(+. 1& .@(></\))@(' '&~:)	Delete extra blanks
d150=: ,.	Join along last dimension
d151=: ,.&.>/	Join items of boxed array

4C. Test

d0=: -:&(/:~)	Are x and y permutations of each other?
m1=: /:~ -:i.@#	Is y a permutation vector?
m2=: -:~@ :	Is y antisymmetric?
m3=: -: :	Is y symmetric?
m4=: [:+./[:*./]=/0 1"_	Are all atoms of Boolean list y equal?
m5=: *./ .= +./	Are all atoms of Boolean list y equal?
m6=: *./ .= *./	Are all atoms of Boolean list y equal?
d7=: -.@(: <:/ .>: >.@] , [)"1	Is y in the half open on the right interval x and is it an integer?
d8=: e.	Is list x a row of array y?
m9=: *./@(>./\)	Are columns of y in ascending order?
m10=: *./@(<./\)	Are columns of y in descending order?
m11=: >./=<./	Are atoms of numerical list y equal?
m12=: *./ +. -.@(+./)	Are atoms of Boolean list y equal?
m13=: *./ = +./	Are atoms of Boolean list y equal?
m14=: *./@({.)	Are atoms of list y equal?
m15=: 0:=# + /	Are atoms of Boolean list y equal?
m16=: *./@(#1& .)	Are atoms of Boolean list y equal?
m17=: ([:,:0:,#)-: v19"_ rxmatch]	Is y a legal J name?
d18=: rxmatch=: 17!:0	" to get rxmatch
v19=: '[:alpha:]][[:alnum:]]_* ' "_	'regex' to get alpha: and alnum:

5A. Structural

The verbs given here affect the structure of the arguments; the shape or rank of the result of such verbs differ from the shape or rank of the argument.

m0=: , :	Add leading dimension of length 1
m1=: , : "_1	Add leading dim of length 1 to items
m2=: , : "1	Add leading dim of length 1 to rows
m3=: , : "0	Add leading dim of length 1 to scalars
m4=: , .	1-column table from list
d5=([: " : [: , . [] , "1 (' ' "_ , "1 [: " :])	Append row stub x to table y
m6=: , :-	Array and its negative

5B. Partition

```

      x=:1 1 0 0 0 1 0 0 1 1
      y=:3 4 8 2 5 6 9 4 5 4
      x +/ ;. 1 y
3 19 19 5 4

```

```

      x < ;. 1 y
+-----+
|3|4 8 2 5|6 9 4|5|4|
+-----+

```

```

      x < ;. 2 y
+-----+
|3|4|8 2 5 6|9 4 5|4|
+-----+

```

```

      x +/ ;. 2 y
3 4 21 18 4

```

The foregoing expressions illustrate the use of the cut conjunction (`;.`) to apply the functions `sum (+/)` and `box (<)` over *partitions* or *fields* of the right argument `y` demarked by the boolean left argument `x`. The case of the box gives a clear picture of the partitioning performed; in case 1, the ones in the left argument mark the beginnings of fields, and in case 2 they mark the ends.

A function (such as the sum scan) that produces non-scalar results illustrates the fact that the box of such a function provides a more readable result:

```

      x +/\ ;. 2 y
3  0  0  0
4  0  0  0
8 10 15 21
9 13 18  0
4  0  0  0

      x <@(+/\) ;. 2 y
+-----+
|3|4|8 10 15 21|9 13 18|4|

```

+-----+

```

      ; x <@(+/\) ;. 2 y
3 4 8 10 15 21 9 13 18 4

```

We therefore define a corresponding conjunction:

```

      cut=: 2 : ' ;@(<@x.;.y.) '
      x +/\ cut 1 y
3 4 12 14 19 6 15 19 5 4

```

```

      x +/\ cut 2 y
3 4 8 10 15 21 9 13 18 4

```

c0=: cut=: 2 : ' ;@(<@x.;.y.) '	
a1=: c1=: cut 1	Case 1 of cut
a2=: c2=: cut 2	Case 2 of cut
d3=: pmax=: >./ c1	Partitioned max over (case 1)
d4=: pmax2=: >./c2	Partitioned max over (case 2)
d5=: pmaxs=: >./\ c1	Partitioned max scan
d6=: pnub=: ~. c1	Partitioned nub
d7=: psort=: /:~ c1	Partitioned sort
d8=: prev=: . c1	Partitioned reverse
m9=: < ;._2@(<,&' : ');._2	UNIX /etc/passwd relation

```

      x
1 1 0 0 0 1 0 0 1 1

```

```

      y
3 4 8 2 5 6 9 4 5 4

```

```

      (x pmax y) ,: (x pmax2 y)
3 8 9 5 4
3 4 8 9 4

```

```

      x([ , ] ,psort ,: prev)y
1 1 0 0 0 1 0 0 1 1
3 4 8 2 5 6 9 4 5 4
3 2 4 5 8 4 6 9 5 4

```

```
3 5 2 8 4 4 9 6 5 4
```

```
p=: >::'sparkle out among the fern to bicker down a valley'
x (,.@[ ; ,.@] ; psort ; prev) p
+-----+
|1|sparkle|sparkle|sparkle|
|1|out    |among   |fern   |
|0|among   |fern    |the    |
|0|the     |out     |among  |
|0|fern    |the     |out    |
|1|to      |bicker  |down   |
|0|bicker  |down    |bicker |
|0|down    |to      |to     |
|1|a       |a       |a      |
|1|valley  |valley  |valley |
+-----+
x  p      psort  prev
```

The monadic case of the 1-cut partitions at each occurrence of the leading item of the argument. Moreover a negative case suppresses the partitioning item. For example:

```
q=: 0 4 2 3 0 4 7 6 0 2
< c1 q
+-----+
|0 4 2 3|0 4 7 6|0 2|
+-----+

psort q
0 2 3 4 0 4 6 7 0 2
```

```
r=: >::'/do you love me / or do you not / you told me once /
but I forgot'
< cut 1 r
+-----+
|/      |/      |/      |/      |
|do     |or     |you    |but    |
|you    |do     |told   |I      |
|love   |you    |me     |forgot |
|me     |not    |once   |       |
+-----+
```

< cut _1 r

do	or	you	but	
you	do	told	I	
love	you	me	forgot	
me	not	once		

5C. Special Matrices and Lists

Although these phrases are intended primarily for numeric matrices, many apply as well to arrays of characters, and to arrays of rank other than 2. For example:

```
m3=: 1: j. # ;. _1
u=: 1 0 0 0 1 1 0 1
m3 u
1j3 1 1j1 1
```

```
d4=: m3@[ # ]
u d4 'abcd'
a bc d
```

```
u d4 i.4 5
0 1 2 3 4
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
5 6 7 8 9
10 11 12 13 14
0 0 0 0 0
15 16 17 18 19
```

```
]cm=: 3 3$'ABCDEFGHI'
ABC
DEF
GHI
```

```
<"2 d5 cm
+-----+
|EF|DF|DE|
|HI|GI|GH|
+---+---+---|
|BC|AC|AB|
|HI|GI|GH|
+---+---+---|
|BC|AC|AB|
|EF|DF|DE|
+-----+
```

```
m42 cm
```

```

+-----+
|ADG|BEH|CFI|
+-----+

(>@m42 ; |:) cm Open of boxed columns is the transpose
+-----+
|ADG|ADG|
|BEH|BEH|
|CFI|CFI|
+-----+

(] ; 0&d12 ; _1 0 1&d12) m=: i. 4 4 Band matrices
+-----+
| 0 1 2 3|0 0 0 0|0 1 0 0|
| 4 5 6 7|0 5 0 0|4 5 6 0|
| 8 9 10 11|0 0 10 0|0 9 10 11|
|12 13 14 15|0 0 0 15|0 0 14 15|
+-----+

```

Several equivalent phrases are provided for a number of cases to illustrate various approaches to the same problem. For example, `m13` uses self-classification to produce an identity matrix, and `m14` uses an equals table. No one solution should be considered to be the "best". If space or time required in execution is of prime concern, then the tools provided in Section 14A (Execution Time and Space) should be used to evaluate the different methods.

<code>d0 =: \$,</code>	Reshape as in APL
<code>d1 =: \$,:</code>	<code>x</code> copies of <code>y</code>
<code>d2 =: -@[[\]</code>	<code>x</code> -column matrix of the items of <code>y</code>
<code>m3 =: 1: j. # ;. _1</code>	Real 1; Imaginary # of 0's between 1's
<code>d4 =: m3@[#]</code>	Expand
<code>d5 =: 1&(:\.)"2^:2</code>	Minors [5, s.v. 'Outfix'] p55 (e.g. <code><"2 d5 i. 3 3</code>)
<code>d6 =: <&.>&.>@{@(i.&.>"_)@ \$ {]</code>	Minors [4, sect. 3.3] p55
<code>m7 =: (<0 1)& :</code>	Diagonal
<code>m8 =: i.@# { "_1]</code>	Diagonal
<code>m9 =: =@i.@# #&,]</code>	Diagonal (tables only)
<code>m10=: <"1@("0~)@i.@# {]</code>	Diagonal
<code>d11=: ir=: i.@#@[</code>	Indices of items of right argument

d12=: [(]*+./@((,@["0 1 ir)=/ir))]	Band matrix (1 0 _1 d12 i. 5 5)
m13=: m14@#	Identity matrix of order of # of items of y
m14=: id=: =@i.	Identity matrix of order y
m15=: =/~@i.	"
m16=: ,~ \$ {.&1@>:	"
m17=: -@>:@i. {"0 1:	"
m18=: -@i. ."0 1 {.&1	"
m19=: ,&1@(\$&0)"0@i.	"
m20=: 0& , ^:(i.`1:)	" (but not for y=0)
m21=: #:@(2&^>@i.@-	"
m22=: [D.1@i.	"
m23=: slt=: >/~@i.	Strict lower triangle
m24=: lt=: >:/~@i.	Lower triangle
m25=: [\@(\$&1)	"
m26=: +./\@=@i.	"
m27=: >:@i. \$"0 1:	"
m28=: >:@i. {"0 1 \$&1	"
m29=: i.@- }."0 1 \$&1	"
m30=: 1& , ^:(i.`1:)	" (but not for y=0)
m31=: i.@- .!.0"0 1 \$&1	"
m32=: #:@(+/\.)@(2&^>@i.@-	"
m33=: ut=: -.@slt	Upper triangle
m34=: +./\"1@=@i.	" but not for y=0
m35=: .!.0^:(i.`(\$&1))	" but not for y=0
m36=: -@i. .!.0"0 1 \$&1	"
m37=: #:@(+/\.)@(2&^>@i.@-	"
m38=: [:%. ,~ \$ {.&1 _1@>:	"
m39=:]\ %@(i.&.<:@+:)	Hilbert matrix but not for y=0
a40=: lor=: [^:	1 lor gives left; 0 lor gives right
m41=: -.@(' '&E.) #]	Remove multiple blanks
m42=: <"1@ :	Box each column of a matrix
m43=: i.@{:@\$ <@:({"1)"0 _]	"

m44=: [:, (i.@2: ,: #,1:)<@,;.3]	"
d45=: <@ :;.1 :	Box columns
n46=: 1 0 1 0 0 1 d45 i.3 6	Box columns 0 1, 2 3 4, and 5
v47=: {.;}.	Split first from rest; split y at x
d48=: i.~ ({.;;).@}.)]	Split y at first x, eliding first x
m49=: }::{:	Split rest from last
c50=: 2 : 'x.@{. , y.@}.'	f c50 g applies f to first item; g to others
m51=: */~@i.	Multiplication table of order y
m52=: >./~@i.	Maximum table of order y
m53=:]]\ ([: <: +:) {. (- {. 1:)	Counterdiagonal matrix of order y
m54=: [: */.@, lt@# >: lnz	Is y lower triangular?
m55=: [: -. 0: e. [: ,]	Is y zero-free?
m56=: lnz=:] ~: 0:	Locate nonzeros in y
m57=: [: */.@, ut@# >: lnz	Is y upper triangular?
m58=: ~:/~@i.	Nondiagonal matrix of order y
d59=: ;@([+&.> <@i."0@])	i{x + i. i{y
m60=: -/\@i.	Alternating series length y
m61=: +/\@i.	First y triangular numbers

6A. Sorting

Sorting means the rearrangement of an array into an order based on some key, which may be either implicit or explicit. The implicit keys are the integer or real number fields, and the standard alphabet. Explicit keys may be any arbitrary set. Note that sorting applies equally well to character data and to Boolean, integer, real numbers, complex numbers, or to boxed data. When applied to real numbers, comparisons are exact. No tolerance is used. Verbs defined to sort in ascending order can be converted to give descending order by changing `/:` to `\:`, as in `m0` and `m1`. Note also that these verbs apply (unless they use the rank operator to prevent it) to arrays of arbitrary ranks, where the items are treated as if they were raveled.

<code>m0=: /:~</code>	Sort the array y in ascending order
<code>m1=: \:~</code>	Sort the array y in descending order
<code>m2=: /:~"_1</code>	Sort the items of array y ascending
<code>d3=: /:@:{ { [</code>	Sort indices x according to y
<code>d4=:]/:{"1</code>	Sort table y according to column x
<code>d5=: \:@[`(/:@[) @.]</code>	Grade x up if y is 1 and down if y is 0
<code>d6=: \:~@[`(/:~@[) @.]</code>	Sort up or down (Try literal argument)
<code>d7=: /:~</code>	Sort y according to x

```

y=: 'leap', 'note', 'file'
m1 y
note
leap
file

m1"1 y
plea
tone
life

z=:> ;: 'to be or not to be that is the question'

( ] ; m0 ; m1 ; m2) z
+-----+
|to      |be      |to      |      |ot|
|be      |be      |to      |      |be|

```

```

|or      |is      |the      |      |or|
|not     |not     |that     |      |not|
|to      |or      |question |      |ot|
|be      |question|or       |      |be|
|that    |that    |not      |      |ahtt|
|is      |the     |is       |      |is|
|the     |to      |be       |      |eht|
|question|to      |be       |      |einoqstu|
+-----+

```

```

q=: ?.15#9
(] , m0 ,: m1) q
1 6 4 4 1 0 6 6 8 3 4 7 0 0 4
0 0 0 1 1 3 4 4 4 4 6 6 6 7 8
8 7 6 6 6 4 4 4 4 3 1 1 0 0 0

```

```

x=: 2 3 4 5 0 1 9 8 7 6
y=:100+10 4 8 6 7 9 5 3 2 1
x ([ , ] ,: d3) y
2   3   4   5   0   1   9   8   7   6
110 104 108 106 107 109 105 103 102 101
9   8   7   1   6   3   4   2   5   0

```

```

y=: ?.5 6$100
d4=: ]/:{"1
(0&d4 ; 2&d4 ; 5&d4) y
+-----+
| 3   5 52 67  0 38|13 75 45 53 21  4|13 75 45 53 21  4|
| 6 41 68 58 93 84| 3   5 52 67  0 38| 3   5 52 67  0 38|
|13 75 45 53 21  4|52  9 65 41 70 91|67 67 93 38 51 83|
|52  9 65 41 70 91| 6 41 68 58 93 84| 6 41 68 58 93 84|
|67 67 93 38 51 83|67 67 93 38 51 83|52  9 65 41 70 91|
+-----+

```

The random number generator `?.` used here produces repeatable experiments, but `?` can also be used. For example:

```

? 9 # 9
1 6 4 4 1 0 6 6 8
?. 9 # 9
1 6 4 4 1 0 6 6 8

```

6B. Ranking & Classification

Ranking is the assignment of an ordinal number to each item of an array, and equal values are given the same rank, with the next rank assigned displaced by the number of ties. Ranking may be exemplified by considering a list of scores in some game. In golf or the card game of hearts a low score is better than a high score. In bowling or gin rummy a high score is better. In a contest where two or more players have equal scores they are ranked equally, but players with the next score are separated in rank by the number of players having equal rank.

m0=: i.~/:~	Rank y rising, 0-origin, ties equal
m1=: >:@m0	Rank y rising, 1-origin, ties equal
m2=: i.~\:~	Rank y falling, 0-origin, ties equal
m3=: >:@m2	Rank y falling, 1-origin, ties equal
m4=: /:@/:	Rank y rising, 0-origin, distinct ranks
m5=: /:@\:	Rank y falling, 0-origin, distinct ranks
m6=: - <./	Normalize y so that minimum atom is 0
d7=:] * [% [: >./]	Scale y by x%max y
d8=: <:@[<.[<.@d7 m6@]	Classify y into x equal intervals
d9=: [:+/"1 ([:i. [:{.]) =/ ([:<. ([- 1: {]) % ([:{:]))	Classify x into {y classes, minimum 1{y, width {:y
d10=: contour=: d12 d11]	Classify altitudes x into contour levels y
d11=: ([: <: [: +/] <:/ [) {]	
d12=: [>. <./@]	

```

]z=: ( ],m0,m1,m2,m3,m4,m5,:m6) y=: 4 2 1 1 2 0 2 5 5 0
4 2 1 1 2 0 2 5 5 0 y
7 4 2 2 4 0 4 8 8 0 m0
8 5 3 3 5 1 5 9 9 1 m1
2 3 6 6 3 8 3 0 0 8 m2
3 4 7 7 4 9 4 1 1 9 m3
7 4 2 3 5 0 6 8 9 1 m4
2 3 6 7 4 8 5 0 1 9 m5
4 2 1 1 2 0 2 5 5 0 m6

```

0.2 " : 10 d7"1 z

Scale

8.00	4.00	2.00	2.00	4.00	0.00	4.00	10.00	10.00	0.00
8.75	5.00	2.50	2.50	5.00	0.00	5.00	10.00	10.00	0.00
8.89	5.56	3.33	3.33	5.56	1.11	5.56	10.00	10.00	1.11
2.50	3.75	7.50	7.50	3.75	10.00	3.75	0.00	0.00	10.00
3.33	4.44	7.78	7.78	4.44	10.00	4.44	1.11	1.11	10.00
7.78	4.44	2.22	3.33	5.56	0.00	6.67	8.89	10.00	1.11
2.22	3.33	6.67	7.78	4.44	8.89	5.56	0.00	1.11	10.00
8.00	4.00	2.00	2.00	4.00	0.00	4.00	10.00	10.00	0.00

x=: 3 _3 10 22 _8

y=: _5 0 5 10 20 30

x ([, d12 , d11 ,: contour) y

3 _3 10 22 _8 x

3 _3 10 22 _5 x increased to not less than min contour level

0 _5 10 20 30 _8 mis-classified because less than lowest contour

0 _5 10 20 _5 Contour

6C. Grading

Grading is the assignment of an ordinal number to each item of an array, with all numbers distinct, and with the earliest of equal values being assigned a lower ordinal number. Grading can be used to produce the permutation that will put an array in order. Since sorting is a primitive operation in J, grading is more commonly used to put a companion array into an order derived from another array. For example, a table of addresses may be put into an order derived from a table of last names.

m0=: /:	Grade y up
m1=: \:	Grade y down
d2=: /:@i.	Grade y up according to key x
d3=: \:@i.	Grade y down according to key x

```

y=: 'leap','note',: 'file'
(] ; m0 ; (m0 { ]) ; m1 ; (m1 { ])) y
+-----+
|leap|2 0 1|file|1 0 2|note|
|note|      |leap|      |leap|
|file|      |note|      |file|
+-----+

x=: ' abcdefghijklmnopqrstuvwxyz' [ y=: 'senator'
x (d2 ; (d2{]) ; d3 ; (d3{])) y
+-----+
|3 1 2 5 6 0 4|aenorst|4 0 6 5 2 1 3|tsronea|
+-----+

```

7A. Permutations

Any re-ordering or *permutation* of the integers $1..n$ is called a *permutation vector* of order n . If p is a permutation vector, then $p\&\{$ is the corresponding permutation function. For example:

```
p=: 4 5 2 1 0 3
text=: 'ABCDEF'
p { text
EFCBAD
```

```
p&{ text
EFCBAD
```

The phrase $k=: A. p$ gives the index of the permutation vector p in the ordered list of $n!$ permutation vectors of order n , and the function $k\&A.$ is the corresponding permutation function. For example:

```
]k=: A. p
590
```

```
k A. text
EFCBAD
```

```
(i.!3) A. i. 3
0 1 2
0 2 1
1 0 2
1 2 0
2 0 1
2 1 0
```

The phrase $c=: C. p$ gives the (boxed) cycle representation of p , and either $p\&C.$ or $c\&C.$ provide the corresponding permutation function. For example:

```

      ]c=: C. p
+-----+
|2|4 0|5 3 1|
+-----+

      c C. text
EFCBAD

```

In the phrases $p \ C. \ x$ and $c \ C. \ x$, the order of the permutation is determined by the number of items in x , and *abbreviated* vectors p and c may therefore be used unambiguously:

```

      3 1 C. text  Move items to tail
ACEFDB

```

```

      (<3 1) C. text  Interchange items
ADCB EF

```

```

      (<3 1 4) C. text Rotate items
AECBDF

```

The application of $c.$ to any abbreviated representation produces the standard form, and two applications of $c.$ therefore provide the standard form for any representation. For example:

```

      C. 3 1
+-----+
|0|3 1 2|
+-----+

```

```

      C. C. 3 1
0 2 3 1

```

```

      C. C. <3 1
+-----+
|0|2|3 1|
+-----+

```

```

C. C. <3 1 4
+-----+
|0|2|4 3 1|
+-----+

```

m0 =: /:	Inverse permutation vector
m1 =: /:&.C.	Inverse cycle
m2 =: (-: >) :: 0:	Not-a-box test
m3 =: m1`m0 @. m2	Inverse permutation
m4 =: C.^:2	Put permutation into standard form
m5 =: (<0 _1)&C.	Interchange first and last items
m6 =: _1&A.	Reverse
m7 =: 3&A.	Rotate last three to the left
m8 =: 4&A.	Rotate last three right
d9 =: ([: +/[[:![:}.[:i.[) A.]	Rotate last x to the left
d10=: (!@[- !@<:@[) A.]	Rotate last x to the right
m11 =: /:~	Sort up
m12 =: \:~	Sort down
m13 =: ?~	Random permutation of order y
m14=: /:@:?\$~	Random permutation of order y
m15=: ?@! A. i.	Random permutation of order y
d16=: A. i.	x-th permutation of order y
m17=: all=: i.@! A. i.	All permutations of order y
m18=: ,:@i.`([: ,/ 0&,.@ (\$:&.<:){ "2 1 \:"1@=@i.)@.(1&<)	All permutations of order y (recursive definition)
m19=: pow=: {^:([)`(i.@#[])	Permutation x to the power y
m20=: [: {/] \$,:@[Permutation x to the power y
m21=: i.@#[C.~(#>@C.@[])#C.@[Permutation x to the power y
m22=: pow 2&^	Permutation x to the power 2^y
m23=: 3 : ('::' ' {~^:y. x.')	Permutation x to the power 2^y
m24=: {~@]^:([)`[]	Permutation x to the power 2^y
m25=: ord=: *./@(#>"_)@C.	The order of a permutation
m26=: sg=: pow i.@ord	Subgroup generated by permutation y
m27=: [: {/\ ord \$,:	"

m28=: ~.@(,/)@({ "1/~)^:_@(i.@#, :])	" § 4.4 Hui [4] p55
d29=: \:@[{]	Move items located by x to front of y
m30=: 1: .]	Rotate y by 1 to the left (or up)
d31=: !@[* !	Number of perms of y objects x at a time
m32=: (] {~ [: /:] = ' ' " _) " 1	Move all blanks to end of row
d33=: /:@[{]	Move items located by x to end of y
m34=: _1: .]	Rotate y by 1 to the right (or down)
m35=: ~.@(,/)@({ "1/~)^:_@(i.@{ : @\$,])	Subgroup generated by a matrix of permutations
m36=: { "1/~	The group table of a matrix of permutations
m37=: ugt=: ~.@(,/)@({ "1/~)	The unique elements of the permutation group
m38=: pbi=: i.@{ : @\$,]	Preface a matrix of permutations by the identity
m39=: ugt^:_ @ pbi	The subgroup generated by a matrix of permutations (same as m35)

7B. Rotations & Reflections

m0 = :]	Identity
m1 = : m6@m7	Three-o'clock rotation
m2 = : m4@m6	Six-o'clock rotation
m3 = : m4@m7	Nine-o'clock rotation
m4 = : .@]	Horizontal reflection
m5 = : m2@m7	Counterdiagonal reflection
m6 = : . "_1@]	Vertical reflection
m7 = : :@]	Diagonal reflection
d8 = : m0`m1`m2`m3`m4`m5`m6`m7 @. [i d8 y gives mi y (all rotates and reflects)

7C. Parity And Symmetry

Parity and *symmetry* each refer to any one of several related notions, including:

Parity

- a. Integers - odd or even
- b. Permutations - odd if equivalent to an odd number of transpositions of pairs; even otherwise
- c. A function - odd if negation of its argument negates its result; even if the result is unchanged; or neither

Symmetry and skew-symmetry

- a. A matrix is symmetric if equal to its transpose; skew if equal to the negation of its transpose; or neither
- b. function is symmetric if it is invariant under permutations of its argument
- c. A function f is symmetric under g if $f \circ g$ is equivalent to f ; is skew if it is equivalent to $-f$; or neither. Item c of parity concerns symmetry under -

Symmetric and skew-symmetric parts

Each has the specified property, and their sum equals the function from which they are derived. For example, \sinh and \cosh are the odd and even parts of the exponential under negation.

In classifying any entity for parity or symmetry, we will give the result -1 for the odd or skew case, 1 for the even or symmetric case, and 0 for neither.

m0=: Isodd=: 2&	Test if y is an odd integer
m1=: Iseven=: -.@Isodd	Test if y is an even integer
m2=: Isperm=: -: /:@/:	Test if y is a permutation vector
a3=: 1 : 'x. -: x.@(?@!@#A.])'	Necessary condition for symmetry of fn.
m4=: m2*_1:^>/~ ~:/@,@:* </~@i.@#	Classify argument as a permutation
m5=: L=: >/~	Left, centre, right limbs of the fork
m6=: C=: ~:/@,@:*	evaluated in m4. See their use below
m7=: R=: </~@i.@#	to demonstrate design of its definition
m8=: m2*_1:^+/@(<:@#@>@(C.@~.))	Parity of permutation from cycle lengths
m9=: -/@(:-:"2],:-)	Classify matrix (skew, neither, sym)
a10=: skn=: .:-	Skew part with respect to negate

a11=: syn=: ..-	Symmetric "
a12=: skt=: .: :	Skew part with respect to transpose
a13=: syt=: .. :	Symmetric "
m14=: sinh=: 5&o.	Hyperbolic sine
m15=: cosh=: 6&o.	Hyperbolic cosine
n16=: m=: 3 1 4,2 0 5,:1 4 1	A 3-by-3 matrix
d17=: ip=: +/ . *	Inner (matrix) product
m18=: LF=: m&ip	A linear function
m19=: cst=: m4"1@(#: i.)@:(#~)	Complete skew tensor; m4 tests for perm
d20=: cross1=: [ip cst@#[ip]	Generalized cross- product
d21=: cross2=: ((_1: .[]*(1: .[]))-((1: .[])*(_1: . []))	Conventional cross product (not valid for dimension greater than 3)
m22=: det1=: +/ @, @(*// * cst@#)	Determinant in terms of cst
m23=: det2=: -/ . *	Determinant
m24=: length=: m24=: +/&.(*)"1	Length of vector
m25=: arcsin=: _1&o.	Arcsine

m26=: angle=:arcsin@(length@ cross1 % length@[*length@])	Angle between two vectors
m27=: dfr=: 180p_1&*	Degrees from radians

The symmetry of a function may be tested (but not guaranteed) by the adverb a3 :

```
+/a3
+/-: +/(@(?@!@# A. ]))

(+/a3,-/a3,*/a3,>./a3,/:~a3) 3 1 4 2
1 0 1 1 1 Sum, product over, max over, and grade are
      symmetric, but the alternating sum is not.
```

Phrases may be analyzed by isolating and executing phrases that occur within them. Consider, for example, phrases m4-7:

```
p6=: (i.@! A. i.) 3
n6=: 3 3 3#i.6
p6 ; (,. m4"1 p6) ; n6 ; (,. m4"1 n6)
```

```
+-----+
|0 1 2| 1|0 0 0|0|
|0 2 1|_1|0 0 1|0|
|1 0 2|_1|0 0 2|0|
|1 2 0| 1|0 1 0|0|
|2 0 1| 1|0 1 1|0|
|2 1 0|_1|0 1 2|1|
+-----+
```

```
perm=: 3 1 4 2 0
(L;R;(L*R);(L C R)) perm

+-----+
|0 1 0 1 1|0 1 1 1 1|0 1 0 1 1|1|
|0 0 0 0 1|0 0 1 1 1|0 0 0 0 1| |
|1 1 0 1 1|0 0 0 1 1|0 0 0 1 1| |
|0 1 0 0 1|0 0 0 0 1|0 0 0 0 1| |
|0 0 0 0 0|0 0 0 0 0|0 0 0 0 0| |
+-----+
```

These panels show that \mathbb{L} compares every pair of elements of the argument for precedence (to see which must be moved over which), and the upper triangle provided by \mathbb{R} masks out double counting.

Used with various functions such as negate and transpose, the conjunctions \cdot and $\cdot\cdot$ yield adverbs that produce odd and even parts of functions to which they are applied:

```
(^skn,sinh,^syn,cosh,: ^ = ^skn + ^syn) a=:i.6
0 1.1752 3.62686 10.0179 27.2899 74.2032
0 1.1752 3.62686 10.0179 27.2899 74.2032
1 1.54308 3.7622 10.0677 27.3082 74.2099
1 1.54308 3.7622 10.0677 27.3082 74.2099
1      1      1      1      1      1

      |y=: (^t. , ^skn t. , sinh t. , ^syn t. ,: cosh t.) a
1 1 0.5 0.1666667 0.04166667 0.008333333
0 1 0 0.1666667 0 0.008333333
0 1 0 0.1666667 0 0.008333333
1 0 0.5 0 0.04166667 0
1 0 0.5 0 0.04166667 0

      % y
1 1 2 6 24 120
_ 1 _ 6 _ 120
_ 1 _ 6 _ 120
1 _ 2 _ 24 _
1 _ 2 _ 24 _

      !^:_1 % y
0 0 2 3 4 5
_ 0 _ 3 _ 5
_ 0 _ 3 _ 5
0 _ 2 _ 4 _
0 _ 2 _ 4 _

      (LF;LF skt;LF syt;LF skt+LF syt) m
+-----+
|15 19 21|_5.5 _3.5 5|20.5 22.5 16|15 19 21|
|11 22 13|_7.5 _3.5 3|18.5 25.5 10|11 22 13|
|12 5 25| 0.5 _1 3.5|11.5 6 21.5|12 5 25|
+-----+
```

Skew arrays may be used in expressions for orthogonality, as in the vector cross product (orthogonal to the plane defined by its arguments), the curl (orthogonal to the

matrix of partial derivatives), and the determinant. We will illustrate this by the completely skew tensor (cst), whose sign is reversed by the transposition of any pair of axes.

```
v=: a cross1 b [a=: 3 1 4 [ b=: 2 0 5
v;(v ip a,.b);(a angle b);(dfr 0 0 1 angle 0 1 0)
+-----+
|_5 7 2|0 0|0.3274544|90|
+-----+
```

```
(cst@#;*//;(cst@#**//);+/@,@(cst@#**//);det1) m
+-----+
| 0 0 0| 6 24 6|0 0 0|_25|_25| | | |
| 0 0 1| 0 0 0|0 0 0| | |
| 0 _1 0|15 60 15|0 _60 0| | |
| | | | | | | | |
| 0 0 _1| 2 8 2|0 0 _2| | |
| 0 0 0| 0 0 0|0 0 0| | |
| 1 0 0| 5 20 5|5 0 0| | |
| | | | | | | | |
| 0 1 0| 8 32 8|0 32 0| | |
|_1 0 0| 0 0 0|0 0 0| | |
| 0 0 0|20 80 20|0 0 0| | |
+-----+
```

8A. Numbers And Counting

Constants such as π (1p1), half- π (1r2p1), integers expressed in base sixteen (16b2a for 42), complex numbers in terms of real and imaginary parts (2j3), and complex numbers in terms of magnitude and angle in degrees or radians (3ad90 and 3ar1) are commonly needed in programming.

Constant *functions* are also needed, as in odd=: 1:+2:*i... The phrase c"r produces a constant function of rank r for any constant c (numeric, or other such as characters or boxed).

Many other constants, such as the reciprocal of π (1p_1), the multipliers used in converting from radians to degrees (180p_1) and from degrees to radians (1r180p1), the kth prime(p: k), and the list of prime factors of an integer (q: i) are commonly used.

The following phrases illustrate the production of various numbers, including such lists and tables as the binomial coefficients, various grids, Stirling numbers, primes and prime factors:

n0 =: pi=: 1p1	π
v1 =: PI=: 1p1"	Other ranks are possible
n2 =: 2p1 1r2p1 1r6p1	List of multiples of π
n3 =: 2p_1 1r2p_1 1r6p_1	Multiples of reciprocal π
n4 =: 1x1	Euler's number (2.71828 ...)
v5 =: 1x1 1x_1 _1x_1"0	List function of rank zero
m6 =: pitimes=: 1p1&*	Like o. but infinite rank
m7 =: ln=: 1x1&^.	Like ^. (natural log)
m8 =: ln=: 1x1"_ ^.]	"
m9 =: [:^ 0j2p1&% * i.	Roots of unity; e.g. m9 5
m10=: 1:=1:^m9	Roots of unity; e.g. m10 5
m11=: sbase=: %:@(2p1&%) * %&1x1 ^]	Stirling's approximation, base term

m12=: scorr=: 1 1r12 1r288 _139r51840 _571r2488320&p. @%	Stirling's approximation, correction
m13=: stirg =: sbase * scorr	Stirling's approx for $\Gamma(y)$ [AS[1] p55 6.1.37]
m14=: 1-(!10)%stirg 1+10	Relative error for !10
m15=: stirf =: ^@(1r12&%) *%:@(2p1&*) * %&1x1 ^]	Stirling's approx to !y [AS[1] p55 6.1.38]
m16=: 1-(!10)%stirf 10	Relative error for !10
m17=: even=: 2:*i.	Even integers
m18=: odd=: >:@even	Odd integers
m19=: odd=: 1:+2:*i.	Odd integers
m20=: grid=: +'(*i.) /	grid b,s,n is From b step s for n
m21=: Ai=: >:@i.	Augmented integers (1 to y)
m22=: +/\@(\$&1)	" (but not negint arg)
m23=: Ei=: i.@>:	Extended integers (0 to y)
m24=: Si=: Ei@+: -]	Symmetric integers (-y to y)
m25=: bc=: Ei !]	Binomial coefficients
m26=: (0&,+,&0)^:([`1:]	"
m27=: bct=: Ei !/ Ei	Pascal's triangle (bc table)
m28=: !/~@Ei	"
m29=: !~/~@Ei	Pascal's triangle
m30=: (0&,+,&0)^:(i.`1:]	"
m31=: 1 1&([: +//. */)^:(i.`1:]	"
m32=: +/\@(.!1.0)^:(i.`(\$&1))	"
m33=: odometer=: #: i.@(* /)	All #s in radix y (odometer)
m34=: >@, @ { @ (i. & . > " _)	" Try m34 10 10
m35=: , : @ i. @ 0 : ` ([: , / i. @ { . , . " 0 2\$: @ } .) @ . (* @ #)	" Try m35 2 2 2
m36=: tt=: #: @ i. @ (2 ^)	Truth table of order y
m37=: odometer@(\$&2)	"
m38=: >@, @ { @ (\$ & (< 0 1))	"
m39=: , : @ i. ` ([: , / i. @ 2 : , . " 0 2 \$: @ < :) @ . *	"

m40=: [: : 2 & ^ (\$ # & 0 1) " (0) 2 & ^ @ i . @ -	"
m41=: (i . @ # = i . ~) #]	Nub (all distinct items of) ~.
m42=: ~ : #]	" Try m42 2 7 1 8 2 8
m43=: + . / @ (< / \ " 1 " _) @ = #]	" Try m43 ; : 'all in all'
m44=: ~ . @ (i . ~) {]	"
m45=: { . / . ~	"
m46=: ({ . ; #) / . ~	Nub and count
m47=: # / . ~	" Try m47 2 7 1 8 2
m48=: + / " 1 @ =	" Try m48 ; : 'all in all'
m49=: 1 : # . =	"
m50=: ifb=: # i . @ #	Index from boolean
d51=: bfi=: i . @ [e .]	Boolean from index; e.g. 0 d51 5 7
d52=: 1 : `] ` (\$ & 0 @ [] }	Boolean from index
m53=: cfpv=: # ; . 1	Count from partition vector
m54=: pvfc=: [: ; { . & 1 & . >	Partition vector from count
m55=: i . @ (+ /) e . 0 & , @ (+ / \)	"
n56=: 2147483647=p: 105097564	The 105,097,564-th prime (0 origin)
m57=: lp=: p : @ i .	List first y primes
m58=: fotp=: (2 & (_ 2 : = - / \) # } :) @ lp	First of twin primes among first y
m59=: p : ^ : _ 1	Number of primes less than y
n60=: 2 2 2 3 5 -: q: 120	Prime factorization of 120
m61=: taut=: -: * / @ q:	y = product of factors. Try m61 12345
m62=: dpe=: (~ . , : 1 : # . =) @ q:	Distinct primes with exponents
m63=: : @ (({ . , #) / . ~) @ q:	"
m64=: taut=: = * / @ (^ /) @ dpe	y = product over powers. Try m64 120
m65=: = * / . ^ / @ dpe	"
m66=: divisors=: / : ~ @ , @ > @ (* / & . > /) @ ((^ i . @ > :) & . > /) @ (_ & q :)	Divisors of y. Try m66 900
m67=: / : ~ @ ~ . @ (* / . ^ " 1 tt@#) @ q:	"
m68=: / : ~ @ , @ : (* / & >) @ { @ (< @ (1 & ,) @ (* / \) / . ~) @ q:	"

m69=: >:@i. ([#~ 0: =)]	"
m70=: perfect=: +: = +/@divisors	Perfect number test
m71=: 1: = #@q:	Prime test
d72=: q:@[-: -.&q:	Relatively prime test
d73=: 1: = +.	"
m74=: totient=:* -. @%~. &. q:	Totient [GKP[3] p55 4.53]
m75=: */@(^/- (^<:))/@dpe	"
m76=: [: +/ 1: = (+.i.)	"
m77=: [:/:~[:~.] [:*[:m50]m73[:i.]	Quadratic residues
d78=: L=: [:{&_1 1 ~e.(*:@}.@i.)@]	Legendre symbol
d79=: L * L~	Quadratic reciprocity
d80=: cfe=:<.@(([:%1:])^(i.@[']))	x terms of continued fraction expansion of y
m81=: rapprox=: (, % +.)&1"0	Rational approximation to y
m82=: [:>[:m50]m72"0[:>[:i.[:<:]	Numbers prime to y
m83=: 3 : '<.@o.&.(10^y.)&*) 1'	y digits of π
m84=: [: +/ %@!@i.	y digits of e
d85=: 4 : '- :@(+y.&%)^:(>.2^.1>.x.-16) x:%:y.'	x digits of the square root of integer y

The last three phrases illustrate operations on extended precision arguments.
(0j_40 ": y formats y in exponential notation using 40 digits.)

```

0j_40 ": m83 40x
3.1415926535897932384626433832795028841971e0
0j_40 ": m84 40x
2.7182818284590452353602874713526624977572e0
0j_40 ": 40 d85 2
1.4142135623730950488016887242096980785697e0

```

m83 exploits the fact that <.@o. produces extended integer results on extended

integer arguments, and uses scaling to effect the desired computation.

m84 uses the reciprocal factorial power series to compute e , and produces extended precision results if given extended precision arguments.

d85 uses Newton's iteration to compute a rational approximation of the square root of integer y , accurate to x digits. The initial estimate is $x \% y$, a rational approximation of the finite-precision square root of y . The number of iterations is the base-2 log of $x-16$, based on that the number of accurate digits doubles on every iteration and that the initial estimate has 16-digits of accuracy.

8B. Grids

m0=: i.	First y non-negative integers
m1=: Ai=: >:@i.	Augmented integers (begin at 1)
m2=: Ei=: i.@>:	Extended integers (0 to y inclusive)
m3=: Si=: -Ei@+:	Symmetric integers (-y to y inclusive)
m4=: grid=: +`(*i.) /	General grid (2{y steps from 0{y by 1{y)
m5=: cb=:] {. 1 _1" _ \$~ >: - 2&	Checkerboard. Try box cb 2 3 4 5
m6=: box=: <"2	Box rank 2 cells (for readable display)
m7=: CB=: *cb@\$	Alternate signs of atoms of y.
m8=: bcb=: _1&=@cb	Boolean checkerboard
m9=: {. 0 1" _ \$~] + 2& @>:	"
m10=: _1: ^ m9	Checkerboard
a11=: syft=: (/~) (@Si)	Symmetric function tables
v12=: WRAP=: wrap : (wrap@]^:[)	Wrap (monad) and repeated wrap (dyad)
m13=: wrap=: RL@(. (next+i.@#)) ^:4	Wrap argument with successive integers
m14=: RL=: .@ :	Roll table to left
m15=: next=: >:@(>./)@,	Next integer after largest in table

The first four phrases give commonly useful integer lists, and the fifth allows specification of the beginning value, the step size, and the number of steps. For example:

```
(m0 ; Ai ; Ei ; Si) 3
+-----+
| 0 1 2 | 1 2 3 | 0 1 2 3 | 3 2 1 0 _1 _2 _3 |
+-----+

(*/~ ; !/~) Si 3
+-----+
| 9 6 3 0 _3 _6 _9 | 1 0 0 0 _1 _4 _10 |
| 6 4 2 0 _2 _4 _6 | 3 1 0 0 1 3 6 |
```

	3	2	1	0	_1	_2	_3		3	2	1	0	_1	_2	_3	
	0	0	0	0	0	0	0		1	1	1	1	1	1	1	
	_3	_2	_1	0	1	2	3		0	0	0	0	1	0	0	
	_6	_4	_2	0	2	4	6		0	0	0	0	_1	1	0	
	_9	_6	_3	0	3	6	9		0	0	0	0	1	_2	1	

```

      grid 3 4 5
3 7 11 15 19

```

Phrases 5-10 give various checkerboard patterns. Note the alternation along each axis:

```

      box cb 2 3 4 5
+-----+
| 1 _1 1 _1 1 | _1 1 _1 1 _1 | 1 _1 1 _1 1 |
| _1 1 _1 1 _1 | 1 _1 1 _1 1 | _1 1 _1 1 _1 |
| 1 _1 1 _1 1 | _1 1 _1 1 _1 | 1 _1 1 _1 1 |
| _1 1 _1 1 _1 | 1 _1 1 _1 1 | _1 1 _1 1 _1 |
+-----+
| _1 1 _1 1 _1 | 1 _1 1 _1 1 | _1 1 _1 1 _1 |
| 1 _1 1 _1 1 | _1 1 _1 1 _1 | 1 _1 1 _1 1 |
| _1 1 _1 1 _1 | 1 _1 1 _1 1 | _1 1 _1 1 _1 |
| 1 _1 1 _1 1 | _1 1 _1 1 _1 | 1 _1 1 _1 1 |
+-----+
      CB i. 3 4
0 _1 2 _3
_4 5 _6 7
8 _9 10 _11

```

Phrase 11 is an adverb that produces function tables on symmetric arguments:

```

      (+syft ; *syft ; !syft) 3
+-----+
| 6 5 4 3 2 1 0 | 9 6 3 0 _3 _6 _9 | 1 0 0 0 _1 _4 _10 |
| 5 4 3 2 1 0 _1 | 6 4 2 0 _2 _4 _6 | 3 1 0 0 1 3 6 |
| 4 3 2 1 0 _1 _2 | 3 2 1 0 _1 _2 _3 | 3 2 1 0 _1 _2 _3 |
| 3 2 1 0 _1 _2 _3 | 0 0 0 0 0 0 0 | 1 1 1 1 1 1 1 |
| 2 1 0 _1 _2 _3 _4 | _3 _2 _1 0 1 2 3 | 0 0 0 0 1 0 0 |
| 1 0 _1 _2 _3 _4 _5 | _6 _4 _2 0 2 4 6 | 0 0 0 0 _1 1 0 |
| 0 _1 _2 _3 _4 _5 _6 | _9 _6 _3 0 3 6 9 | 0 0 0 0 1 _2 1 |
+-----+

```

The final phrases produce spirals by wrapping a table (or by default a list or scalar) in a blanket of successive integers:

```

      ]t=: i. 2 3
0 1 2
3 4 5

      (RL t) ; (WRAP 0) ; (3 WRAP 0)
+-----+
| 2 5 | 6 7 8 | 42 43 44 45 46 47 48 |
| 1 4 | 5 0 1 | 41 20 21 22 23 24 25 |
| 0 3 | 4 3 2 | 40 19  6  7  8  9 26 |
|     |       | 39 18  5  0  1 10 27 |
|     |       | 38 17  4  3  2 11 28 |
|     |       | 37 16 15 14 13 12 29 |
|     |       | 36 35 34 33 32 31 30 |
+-----+

```

8C. Representations

d0=: >:@<.@^.	Length to represent y in base x
d1=: d0 # [Base to represent integer y in base x
d2=: d1 #:]	Base x rep of decimal integer y
m3=: #:~i.@>:	Binary reps of integers from 0 to y
m4=: 10"_ d2]	Base 10 rep of decimal integer y
m5=: 16"_ d2]	Base 16 rep of decimal integer y
d6=: (>:@<.@^.# [) #:]	Base x rep of decimal integer y
d7=:] #:~>:@<.@^.# [Base x rep of decimal integer y
m8=: 100"_ d2]	Base 100 rep of decimal integer y
m9=: 0 1&#:	Whole and fractional parts of y
m10=: 10 100 1000"_ #:]	Mixed base 10 ^ 1 2 3 rep of y
m11=: #.	Integer from Boolean list y
m12=: <.@(10&^.)@((' .' &~: #]) &." : %])	Number of decimal places in y
m14=:]&__	Negative infinity
m15=: _:	Positive infinity
a16=: (&#.) (^:_1)	Base x rep of decimal integer y
m17=: [: (% "1 [: , :~ +. /) , :&1	Fraction equivalent to y

8D. Arithmetic

d0=: [: >: #:	Add 1 to cyclic counter y with upper limit x
d1=: +//.	Sum items of y corresponding to keys x
d2=: +//. {~ 'abcde' "_ i. ~.@[Sum y according to keys x from key set 'abcde'. (substitute your ordered keyset for 'abcde')
d3=: +.	Greatest common divisor of x and y
m4=: >:@i. @(>./) /]	1 thru max y residues of y
m5=: 0: = m4	Locate positive integers <: y dividing y
d6=: +"1	Add x to each row of y
d7=: +	Add list x to each column of table y
d8=: +	Add list y to each column of table x
d9=: *.	Least common multiple of x and y

8E. Complex Numbers

m0=: cnj=: +	Conjugate
m1=: mag=:	Magnitude
m2=: %:@(cnj*])	"
m3=: rai=: +.	Real and imaginary parts
m4=: maa=: *.	Magnitude and angle
m5=: irai=: rai^:_1	Inverse rai
m6=: imaa=: maa^:_1	Inverse maa
m7=: rou=: [:^ 0j2p1&% * i.	Roots of unity
m8=: rpg=: rai@rou	Regular polygon
d9=: zero=:] * 10&^@-@[<	Zero any real y less than 10^-x in mag
m10=: z=: ({.,{:*1e_6"_<%~/@:)&.rai	Zero imaginary if relatively small
m11=: (1e_10&\$:) : (j./"1@((] * (<:)) +.))	Clean y

The function z may be used to zero any imaginary part that is relatively small compared to the corresponding real part. For example:

```
(] ,: z) a=:3+j.10^-2*i. 5
3j1 3j0.01 3j0.0001 3j1e_6 3j1e_8
3j1 3j0.01 3j0.0001      3      3

z a
3j1 3j0.01 3j0.0001 3 3
```

Complex numbers can be scaled by multiplication by a real number, and shifted and rotated by addition and multiplication by complex numbers. For example:

```
]a=: rou 3 Third roots of unity
1 _0.5j0.866025 _0.5j_0.866025

|a Lie on the unit circle
1 1 1
```

1ad30 Complex of mag 1 and angle of 30 degrees
0.866025j0.5

```
6&zero@rai&.> (] ; 3j4&+ ; 1ad30&* ; 2ad60&*) a
+-----+
| 1      0| 4      4| 0.866025 0.5| 1  1.73205|
| 0.5 0.866025| 2.5 4.86603|_0.866025 0.5|_2      0|
|_0.5 0.866025| 2.5 3.13397|      0  _1| 1  _1.73205|
+-----+
Coordinates  Shift by 3,4  Rotate by  Rotate by
of triangle           30 degrees  60 degrees
```

9A. Matrix Algebra

m0=: I=: =@i.@#	Identity matrix of order of y
m1=: inv=: %.	Matrix inverse
d2=: mq=: %.	Matrix quotient
d3=: mp=: +/ . *	Matrix product
d4=: amp=: -/ . *	Alternating matrix product
m5=: det=: -/ . *	Determinant
d6=: fit=: {:@] %. {:@] ^/ i.@[d fit x,:f x is coeffs of pn fit of degree d-1

9B. Linear Vector Functions

If $f(x+y)$ equals $(f\ x)+(f\ y)$ for all vectors x and y in its domain, then f is said to be a linear vector function. In other words, $f@:+ -: f@[+ f@]$ is a test for the linearity of f :

```
f=: +/\ "1 Sum scan (subtotals) is a linear vector function
x=: 4 3 2 1 0
y=: 2 3 5 7 11
x ([ , ] , f@[ , f@[ , (f@[ + f@[)) ,: f@:+) y
4 3 2 1 0 x
2 3 5 7 11 y
4 7 9 10 10 f x
2 5 10 17 28 f y
6 12 19 27 38 (f x)+(f y)
6 12 19 27 38 f (x+y)
```

For vector arguments of dimension n , any linear vector function f can be expressed as a matrix product, using a matrix obtained by applying f to the identity matrix of order n . Moreover, the inverse of a linear function is a linear function. For example:

```
g=: f^:_1 Inverse of subtotals is first differences
(] ; f ; g ; f@g) y
+-----+
|2 3 5 7 11|2 5 10 17 28|2 1 2 2 4|2 3 5 7 11|
+-----+

I=: = i. 5 Identity matrix
mf=: (f I) [ mg=: (g I)
mp=: +/ . * Matrix product
mf ; mg ; (y,(y mp mf),:f y) ; (mf mp mg)
+-----+
|1 1 1 1 1|1 _1 0 0 0|2 3 5 7 11|1 0 0 0 0|
|0 1 1 1 1|0 1 _1 0 0|2 5 10 17 28|0 1 0 0 0|
|0 0 1 1 1|0 0 1 _1 0|2 5 10 17 28|0 0 1 0 0|
|0 0 0 1 1|0 0 0 1 _1| |0 0 0 1 0|
|0 0 0 0 1|0 0 0 0 1| |0 0 0 0 1|
+-----+
```

a0=: MR=: 1 : 'x.@=@i.@#'	Matrix representation of linear function
d1=: mp=: +/ . *	Matrix product
a2=: L=: &mp	Linear function represented by matrix
a3=: inv=: ^:_1	Inverse adverb
a4=: MRI=: (^:_1) 'MR' f.	Matrix representation of inverse function
a5=: MR (%. @)	"

```

m=: f MR y [ mi=: f MRI y
m ; mi ; y,(y L m),:(mi L m L y)

```

```

+-----+
|1 1 1 1 1|1 _1 0 0 0|2 3 5 7 11|
|0 1 1 1 1|0 1 _1 0 0|2 5 10 17 28|
|0 0 1 1 1|0 0 1 _1 0|2 5 10 17 28|
|0 0 0 1 1|0 0 0 1 _1|
|0 0 0 0 1|0 0 0 0 1|
+-----+

```

9C. Polynomials and Rational Functions

Sums, differences, products, quotients, derivatives, integrals, and compositions of polynomials can be expressed as functions of their coefficients. For example:

```

c=: 1 4 6 4 1 [ d=: 1 2 1 [ x=: i.6
ppr=: +//. @ (*/)
c ppr d
1 6 15 20 15 6 1

((c ppr d)p. x) ; ((c p. x)*(d p. x))
+-----+
|1 64 729 4096 15625 46656|1 64 729 4096 15625 46656|
+-----+

```

The polynomial function `p.` and related facilities such as the Taylor coefficients `t.` are all defined in terms of ascending powers, as is appropriate to power series. The definition in terms of descending powers commonly used in high-school algebra is discussed at the end of the section.

d0=: sum=: +/@,:	Polynomial sum. Try 1 2 sum 1 3 3 1
d1=: dif=: -/@,:	" difference
d2=: ppr=: +//. @ (*/)	" product
m3=: pdi=: 1: }.] * i.@#	" derivative
d4=: pin=: ,] % >:@i.@#	" integral (left arg gives constant)
m5=: piz=: 0&pin	" integral 0 constant of integration
m6=: atop=: [+/ .* 1 0"_ ,ppr/\@(<:@#[# ,:@])	Composition: (c atop d)&p. is equivalent to (c&p.) @ (d&p.)

Binomial coefficients have an important property illustrated by the following:

```

bc=: i.@>: ! ]
bc n=: 4
1 4 6 4 1
(bc n) p. x=: i. 6
1 16 81 256 625 1296

(x+1) ^ n
1 16 81 256 625 1296

```

This behaviour is extended to the coefficients of a polynomial as follows:

```

bct=: !/~@i.
expand=: bct@# +/ . * ]
]d=: expand c=: 3 1 4 2
10 15 10 2

(c p. x+1) ,: (d p. x)
10 37 96 199 358 585
10 37 96 199 358 585

```

A polynomial with coefficients c may also be expressed as the product over its roots multiplied by the coefficient of the highest-order term, that is, $\{ : c$. The self-inverse monad $p.$ provides the transformations between coefficients and roots. For example:

```

c=: _126 _87 _6 3
]r=: p. c
+-----+
|3|7 _3 _2|
+-----+

p. r
_126 _87 _6 3

(c p. x) ,: (r p. x)
_126 _216 _300 _360 _378 _336
_126 _216 _300 _360 _378 _336

p. 1 2 3

```

```
+-----+
|3|_0.3333333j0.4714045 _0.3333333j_0.4714045|
+-----+
```

m7=: p.	Transformation between roots and coefficients
d8=: p.	Polynomial in terms of roots or coefficients
c9=: FIT=: 2 : 'x. %.]^(i.y.)"_'	f FIT d gives coeffs of pn fit of degree d-1

The *falling factorial* function `ff=: ^!._1`, and the corresponding falling polynomial `fp=: p.!._1` are useful in the finite calculus:

```
ff=: ^!._1
fp=: p.!._1 " 1 0
c=: 2 1 4 3 [ x=: 6 [ n=: 4
(x^n),(*x+0*i.n)
1296 1296

FIT=: 2 : 'x. %. ] ^/ (i.y.)"_'
(x ff n),(*x+_1*i.n)
360 360

(c p. x),(+c*x^i.#c)
800 800

(c fp x),(+c*x ff i.#c)
488 488
```

We will define a linear function to transform the coefficients of a polynomial to the coefficients of an equivalent falling polynomial, that is, `(c p. x)-:(fcFc fp x)`:

```
VM=: ((/ ~) (@i.)) (@#) Vandermonde adverb
^VM Normal Vandermonde
^/~@i.@#

^VM c=: 3 1 4 2
1 0 0 0
1 1 1 1
1 2 4 8
```

1 3 9 27

fcFc=:] +/ . *~ ^VM %. ff VM Falling coeffs from normal
coeffs

fcFc c=: 3 1 4 2
3 7 10 2

(c p. x) ,: (fcFc c) fp x=: 0 1 2 3 4 5
3 10 37 96 199 358
3 10 37 96 199 358

d10=: ff=: ^!._1	Falling factorial (_1-stope)
d11=: fp=: p.!._1 " 1 0	Falling factorial polynomial
a12=: VM=: ((/ ~)(@i.))(@#)	Vandermonde adverb
m13=: fcFc=:] +/ .*~^VM%.ff VM	Falling coeffs From ordinary coeffs
m14=: cFfc=: fcFc^:_1	Ordinary coeffs From falling coeffs
d15=: taut=: p. -: fcFc@[fp]	Tautology
d16=: rf=: ^!.1	Rising factorial
a17=: S=: 1 : '^!.x.'	Stope adverb (1 S is rf Try 0.1 S)
d18=: mtn=: {.@[+/ . *] */ .^ }.@[Multinomial e.g. (c,E) mtn x,y,z

If *c* is a list of coefficients equal in number to the columns of a three-rowed table of exponents *E*, and if *v*=: *x,y,z*, then *c +/ . * v */ . ^ E* is a *multinomial*, a weighted sum of powers of *x*, *y*, and *z*. For example:

```
v=: 'x y z'=: 2 3 5
c=: 3 1 4 2 [ E=: 1 0 2 3,1 1 0 0,:1 2 1 0
E ; (, .v*/ . ^ E) ; (c +/ . * v*/ . ^ E)

+-----+
|1 0 2 3|30|261|
|1 1 0 0|75|   |
|1 2 1 0|20|   |
|      | 8|   |
+-----+

mtn=: {.@[ +/ . * ] */ . ^ }.@[
(c,E) mtn v
261
```

If f and g are polynomials, then the function $f \% g$ is called a rational function. The conjunction $R=: 2 : 'x.\&p. \% y.\&p.'$ produces a rational function defined by its coefficient arguments:

```
R=: 2 : 'x.\&p. \% y.\&p.'
c=: 1 4 6 4 1 [ d=: 1 2 1 x=: i.6
c R d
1 4 6 4 1&p. \% 1 2 1&p.

c R d x
1 4 9 16 25 36

d R c x
1 0.25 0.1111111 0.0625 0.04 0.02777778
```

The Taylor coefficients of rational functions may provide interesting results. For example:

```
c R d t. i. 10
1 2 1 0 0 0 0 0 0 0

d R c t. i. 10
1 _2 3 _4 5 _6 7 _8 9 _10

0 1 R 1 _1 _1 t. i. 10
0 1 1 2 3 5 8 13 21 34 Fibonacci numbers
```

c19=: R=: 2 : 'x.\&p. \% y.\&p.'	Rational conjunction
c20=: TR=: 2 : '(x.\&p. \% y.\&p.) t.'	Taylor series of rational function

The high-school definition of a polynomial in terms of descending powers may be defined by reversing the order of the coefficients as follows:

```
dp=: (|.@[ p. ])"1 1 0
1 2 3 4 p. i. 7
1 10 49 142 313 586 985

4 3 2 1 dp i. 7
1 10 49 142 313 586 985
```

Corresponding definitions of some other functions such as sum, product, and derivative are collected below:

d21=: dp=: (.@[p.])"1 1 0	Polynomial in descending powers
d22=: dsum=: sum& . .	Polynomial sum in descending powers
d23=: ddif=: dif& . .	Polynomial difference in descending powers
d24=: dppr=: ppr	Polynomial product in descending powers
m25=: dpdi=: pdi& . .	Polynomial derivative in descending powers
m26=: ".&> 'a=:2&{' ; 'b=:1&{' ; 'c=:0&{'	Coefficients a, b, and c of quadratic
m27=: dsc=: (b^2:) - 4:*a*c	Discriminant of quadratic
m28=: r=:(-@b(+,-)%:@dsc)%+:@a	Roots of quadratic
d29=: m28@(1: , ,)	b d29 c gives roots of 1,b,c
m30=:] d29"0 i.@>.@(*: % 4:)	Arguments limited to real results

For example:

```

d=: 1 2 3 4 [ e=: 3 2 5
d dsum e
1 5 5 9
((d dsum e)dp y) ,: (d dp y) + (e dp y) [ y=: i.7
9 20 47 96 173 284 435
9 20 47 96 173 284 435

((d dppr e)dp y) ,: (d dp y) * (e dp y)
20 100 546 2204 6832 17460 38750
20 100 546 2204 6832 17460 38750

```

Phrases m33–m35 use the well-known expression from elementary algebra to produce the roots of a quadratic as functions of a three-element list of coefficients for a polynomial with exponents in ascending order. For example:

```

(a ; b ; c ; dsc ; r) abc=: 2 _7 3
+-----+

```

```
|3|_7|2|25|2 0.3333333|
+-----+
```

```
abc dp r abc Test roots
0 0
```

```
(a ; b ; c ; dsc ; r) abc=: 1 1 1
+-----+
|1|1|1|_3|_0.5j0.866025 _0.5j_0.866025|
+-----+
```

```
abc dp r abc
1.11022e_16 1.11022e_16
```

The expression `b d36 c` gives the roots of the "monic" polynomial with coefficients `1, b, c`, and `m36` applies it to pairs of non-negative integers that yield real roots. For example:

```
<"1 (i.6) d36"0/ i.3
+-----+
|0 0 |0j1 0j_1 |0j1.41421 0j_1.41421 |
+-----+
|0 _1|_0.5j0.8660254 _0.5j_0.8660254|_0.5j1.32288 _0.5j_1.32288|
+-----+
|0 _2|_1 _1 |_1j1 _1j_1 |
+-----+
|0 _3|_0.381966 _2.61803 |_1 _2 |
+-----+
|0 _4|_0.2679492 _3.73205 |_0.5857864 _3.41421 |
+-----+
|0 _5|_0.2087122 _4.79129 |_0.4384472 _4.56155 |
+-----+
```

```
m37 6
      0      _6
_0.1715729 _5.82843
_0.3542487 _5.64575
_0.5505103 _5.44949
_0.763932 _5.23607
      _1      _5
_1.26795 _4.73205
_1.58579 _4.41421
      _2      _4
```

9D. Transcendental Functions

m0 =: (^1)&^	Exponential ^
m1 =: (^1)&^.	Natural log ^.
m2 =: antilog=: 10&^	Ten-to-the-power (Base-10 Antilog)
m3 =: log=:10&^.	Base-10 log
m4 =: sin=: 1&o.	Sin
m5 =: cos=: 2&o.	Cos
m6 =: tan=: 3&o.	Tan
m7 =: sinh=: 5&o.	Sinh
m8 =: cosh=: 6&o.	Cosh
m9 =: tanh=: 7&o.	Tanh
m10=: ^ .:-&.j.	Sin as odd part of exponential
m11=: ^@j. ...-	Cos as even part
m12=: ^ .:-	Sinh as odd part
m13=: ^ ...-	Cosh as even part
m14=: sin -: sinh&.j.	Tautology
m15=: cos -: cosh& j.	"
m16=: tan -: tanh&.j.	"
m17=: sinh -: sin&.j.	"
m18=: tan -: tanh&.j.	"
m19=: tanh -: tan&.j.	"
d20=: * -: +&.^.	"
d21=: * -: +&.(10&^.)	"
v22=: % -: -&.^.	"
d23=: + -: *&.^	"
d24=: + -: *&.(10&^)	"
v25=: - -: %&.^	"
m26=: gamma=: !@<:	Gamma function of y
d27=: beta=: *&m26 % m26@+	Beta function of x and y

9E. Quadrature and Simpson's Rule

m0=: W=: integ p. +: >. 1:	Weights for 1+2*y points; try m0 3x
m1=: integ=: pint"1@%.@vm	Integral of inverse of Vandermonde
m2=: vm=: ^~/~@i.@>:@+:	Transposed Vm of order 1+2*y
m3=: pint=: 0: ,] %>:@i.@#	Polynomial integral of coefficients y
m4=: pder=: 1: }.] *i.@#	Polynomial derivative of coefficients y
d5=: EW=: +/@(-@take {."0 1])	Extended weights (x groups of #y points)
d6=: take=:i.@[(] + <:@]* [) #@]	Amounts of overtake

The integral of a function f from a to b is called its *quadrature*. Quadrature can be approximated by a weighted sum of the values of the function on a uniform grid from a to b . Use of the weights $(1 \ 4 \ 1)\%3$ on a grid of three points is equivalent to integrating the parabola (polynomial of order 2) fitted to the function at the points. Use of these weights is referred to as an application of *Simpson's Rule*.

The foregoing phrases (based on the development in [Calculus p55\[6\]](#)) provide the weights for Simpson's Rule and for higher-order polynomial approximations as illustrated below:

```

      ,.(W 1x);(W 2x);(W 3x)
+-----+
|1r3 4r3 1r3|
+-----+
|14r45 64r45 8r15 64r45 14r45|
+-----+
|41r140 54r35 27r140 68r35 27r140 54r35 41r140|
+-----+

      ,.(3*W 1x);(45*W 2x);(140*W 3x)
+-----+
|1 4 1|
+-----+

```

```
| 14 64 24 64 14 |
+-----+
| 41 216 27 272 27 216 41 |
+-----+
```

If weights for n points are applied by dividing the grid into groups of n points, the weights are applied as illustrated below for four groups of Simpson's Rule:

```
(-3 5 7 9) {."0 1 w=: 1 4 1
1 4 1 0 0 0 0 0 0
0 0 1 4 1 0 0 0 0
0 0 0 0 1 4 1 0 0
0 0 0 0 0 0 1 4 1

+ / (-3 5 7 9) {."0 1 w
1 4 2 4 2 4 2 4 1
```

The function `EW` provides such extended weights as illustrated below:

```
3 * 5 EW W 1
1 4 2 4 2 4 2 4 2 4 1

45 * 4 EW W 2
14 64 24 64 28 64 24 64 28 64 24 64 28 64 24 64 14
```

9F. Geometry

We begin with some simple functions (Length, Area, Volume) of various figures (rectangle, box, circle, cone, sphere, pyramid) applied to a length or list of lengths. For example:

m0=: Ar=: */	Area of rectangle
m1=: Ab=: 2: * [:+ /] *1& .	Area of box
m2=: Vb=: */	Volume of box
m3=: Lci=: 2: * o.	Length (circumference) of circle (radius)
m4=: Aci=: [:o.] ^ 2:	Area of circle (r)
d5=: Aco=: o.@*	Area of cone, excluding base (h r)
d6=: Vco=: 1r3p1"_ *] * *	Volume of cone (h r)
m7=: As=: 4p1"_ *] ^ 2:	Area of sphere (r)
m8=: Vs=: 4r3p1"_ *] ^ 3:	Volume of sphere (r)
m9=: L=: +/&.(*:"_)"1	Length of a vector
d10=: Lp=: [: L [, [: L [: -:]	Length of edges of pyramid (h w,l)
d11=: Ap=: [:+ /]* [:L"1 [, "0-:@]	Area of pyramid, excluding base (h w,l)
d12=: Vp=: 1r3"_ * */@,	Volume of pyramid
m13=: sp=: -:@(+ /)	Semi-perimeter
m14=: h=: [: %: [: */ sp 0: ,]	Heron's formula for area of triangle

For example:

```

h 3 4 5
6

h 51 52 53
1170

h 2 2 2
```

1.73205

In treating coordinate geometry we will use a list of n elements to represent a point in n-space, and an m by n table to represent a polygon of m vertices. For example:

```
p=: 3 1 [ q=: 4 1 [ r=: 5 9 Three points
T=: p,q,:r A triangle
L=: +/&.(*:"_) "1 Length function
L p
3.16228

u=: 1&|. Rotate up
D=: u-] Displacements

,.&.>(;u;D;L@D)T3 Displacements and lengths (of sides)
+-----+
| 3 1|4 1| 1 0|          1|
| 4 1|5 9| 1 8|8.06226|
| 5 9|3 1|_2 _8|8.24621|
+-----+

line=: 3 1 4,:1 5 9 A line in 3-space
(;-/;L@(-/);L) line Line, disp, length, distances to ends
+-----+
| 3 1 4|2 _4 _5|6.7082|5.09902 10.3441|
| 1 5 9|          |          |
+-----+

T3=: ?.3 3$10 Random triangle in 3-space
,.&.>(;u;D;L@D) T3
+-----+
| 1 7 4|5 2 0| 4 _5 _4|7.54983|
| 5 2 0|6 6 9| 1 4 9|9.89949|
| 6 6 9|1 7 4|_5 1 _5|7.14143|
+-----+
```

m15 =: L=: +/&.(*:"_) "1	Length
m16=: D=: 1& . -]	Displacement
m17=: LS=: L"1@D	Lengths of sides
m18=: S=: 1&o.@(*&1r180p1)	Sine in degrees
m19=: C=: 2&o.@(*&1r180p1)	Cosine in degrees
m20=: r=: (C,S),:(-@S,C)	2-dim rotation matrix in degrees

m21=: b=: <"1@("0/~)	Table of boxed index pairs: do i 0 2
d22=: R=: (r@])^(b@[)^(=@i.@3:)} a	3-dim rm: From axis 0 to 2 is 0 2 R
d23=: mp=: +/ . *	Matrix product
m24=: R3=: (2 0"_ R 0&{)mp(1 2"_ R 1&{)mp(0 1"_ R 2&{)	R3 p,q,r is p-rotate from axis 2 to 1 on q-r from 1 to 2 on r-r from 0 to 1
m25=: Det=: -/ . *	Determinant
m26=: Area=: [:Det] ,. %@!@{:@\$	Area of triangle
m27=: Vol=: Area f.	Volume of simplex in n-space (fixed)
d28=: dsplitby=: ~:/@:*@:Vol@: (,"1 2)	Are points pairs (2 by n matrix) x separated by n by n simplex y?
m29=: Area2=: [: -: [: +/ 2: Det\]	Area of polygon

Area yields the area of a triangle expressed as a 3 by 2 list of x-y coordinates:

```

      TT
3 3
6 5
2 7

      Area TT
7

```

Area2 also yields the area of a triangle, expressed by a similar table, but with the top row repeated at the bottom:

```

      TT2
3 3
6 5
2 7
3 3

      Area2 TT2
7

```

It is more general, however, and will yield the area of arbitrary polygons:

```

      Polygon

```

7 2
10 5
6 8
3 6
4 3
7 2

Area2 Polygon
24.5

10A. Sums and Means

m0=: A=: # %~ +/	Arithmetic mean
m1=: G=: # %: */	Geometric mean
m2=: H=: A&.(%"_)	Harmonic mean
m3=: C=: {.@((G,A)^:_)	Common mean (de Kerf [2] p55)
m4=: A&(^&3)	Mean of cubes
a5=: (^&) (A&.)	Generalized x-mean
m6=: m4 -: 3 a5	Tautology
a7=: (^&) (A&.)	L-x norm
m8=: (^&3)@(3 a7) -: (3 a5)	Tautology
v9=: MA=: A\	x-period moving average of y
m10=: +/\ -: +/\.&. .	Tautology: invariance under reverse
m11=: +/\ -: +/\ &. .	Tautology: invariance under reverse
m12=: +/\^(i.`(\$&1))	Figurate numbers
m13=: 1: #.]	Sum; +/"1
m14=: +/&.(*)"_)@:+"0	Magnitude
m15=: - .!.0	./\^:_1
d16=: psum=: ([:+/ [^~ i.@])"0	Sum of powers of integers; 2 sum 7
m17=: 0&psum -: [Tautology: +/(i.n)^0
m18=: 1&psum -: 2&!	Tautology: +/(i.n)^1
m19=: 2&psum -: 0 1r6 _1r2 1r3&p.	Tautology: +/(i.n)^2
m20=: [(psum %. ^/~@) i.@(2&+)	Polynomial coeff. of +/(i.n)^k

10B. E Pluribus Unum

m0=: >./	List maximum or column maxima
d1=: >./ .*	Maximum of x weighted by y
d2=: <./ .*	Minimum of x weighted by y
d3=: +/ .= ,	How many x in y?
m4=: +/"1	Sum rows of y
m5=: >./ -<./	1 less than range of y
m6=: [: >./ 0: ,]	Maximum of y, but at least 0
m7=: <./	List minimum or column minima

10C. Math & Stats

m0=: median=: <.@-:@# { /:~	Median of y
m1=: tc=: (+./ .*.~)^:_	Transitive closure of connection matrix y
c2=: ^:	Apply function x y times
c3=: ^:	Conditional application of x if y is 0 or 1
m4=: gamma=: !@<:	Gamma function of y
d5=: beta=: *&m4 % m4@+	Beta function of x and y
d6=: pd=: ^@-@] * ^~ % !@[Poisson distribution of states x with average number y
d7=: outof=: !	# of combs of y objects taken x at a time
d8=: ppr=: +//. @(* /)	Product of polynomials x and y
m9=: -<./ "1	Move rows of table y into first quadrant
m10=: npb=: p:^:_1	Number of primes before y
m11=: pto=: p:@i.@npb	List of primes to y
d12=: dpr=: [: 0 2 1 3& : */	Direct matrix product of x and y
d13=: shurp=: (0 2;3 1)& :@(* /)	Shur product of x and y
d14=: shurs=: (0 2;3 1)& :@(+ /)	Shur sum of x and y
m15=: <./ .+~	Extend distance table to next leg
d16=: +./ .*.	Extend a transitive binary relation
d17=: +/ .* % #@]	Mean of x weighted by y
d18=: -/ .%	Alternating sum of reciprocal series x%y
d19=: +/ .%	Sum of reciprocal series x%y
d20=: +/ .*	Matrix product
d21=: +/ .*	Sum over subsets of x located by y
m22=: +/ .*~	Sum of squares of y
d23=: +/ .*	Scalar (dot) product of vectors x and y
d24=: */ .^	Product over subsets of x located by y

m25=: +/@: *:	Sum of squares of y
m26=: det=: -/ .*	Determinant
m27=: (+/@(*:@(] - +/ % #)) % #)"1	Sample variance (dispersion) of y
m28=: %:@m27"1	Standard deviation of y
d29=: (+/@((]-+/@]%#@])^[]%#@])"1	x-th moment of y
m30=: m31"1	Mean of rows of table y
m31=: +/%#	Mean of list y or columns of table y

10D. Plotting

m0=: # "0&1	Stacks of 1s of length y followed by 0s
m1=: #.&. :	Decimal value of 2-item Boolean list y
m2=:] { ' +ox' "_	One of four characters chosen by y
m3=: m2@m1@m0"1	Horizontal barchart of two intgr series y
m4=: # "0&'x'	Stack of 'x's of length y followed by 's
m5=: [: i. [: >: >./	Dense indices from 0 thru max y
m6=: [: >: [: i. >./	Dense indices from 1 thru max y
d7=: [+ [: i. [: -. -	Dense indices from x thru y
m8=: <./ , >./	Min and max of y
m9=: #/.~	Frequency count of items of y
m10=: <./ d7 >./	Integers from min y to max y, inclusive
m11=: [: >: [: i.]	Integers from 1 thru y, inclusive
m12=: [: i. [: >:]	Integers from 0 through y, inclusive
m13=: m25 d18 'x' "_	Barchart of y on vertical axis using 'x'
m14=: m25 d18 1:	Barchart of y on vertical axis using 0 1
m15=: .@ :@m13	Barchart of y on horizontl axis using 'x'
m16=: .@ :@m14	Barchart of y on horizontl axis using 0 1
m17=: [: -. m20	Table with i{y trailing 0s on row i
d18=: # "0	x replications of y
m19=: .@ :@m23	Barchart of y (up the page)
m20=: [: ."1 m22	Table with i{y trailing 1s on row i
m21=: [: -. m22	Table with i{y leading 0s on row i

m22=: d18&1	y replications of 1, trailing 0s
m23=: d18&'x'	Barchart of y (across the page)
d24=: [{ .] # 1:	List of y 1s followed by x-y 0s
m25=: [: <: [: m9 m11@(>./) ,]	Count of y among 1 through max y
m26=: [: <: [: m9 m12@(>./) ,]	Count of y among 0 through max y
m27=: [: <: [: m9 m10 ,]	Count of y among min through max y
d28=: [:+/\{ .@] ,>:@[#{ } .-} :)@] %>: @ [Interpolate x values between items of y
d29=: \$@]\$((* < .@: % #@]) /: @/ : @ ,)	Classify y into x groups

10E. Approximation

d0=: ^.@] ^@([+/ .* %.) 1: ,. [Exponential fit of x and y
d1=:] ([+/ .* %.) 1: ,. [Linear least squares fit of x and y
d2=: [: ^ 1& ,.@[% .~ ^.@]	a and b such that y is approx. $a * b^x$
d3=: 1& ,.@[% .~]	a and b such that y is approx. $a + b*x$

10F. Random Numbers

d0=: { .@[+ ?@ (] \$ -@ (- /@ ())	Shape y array of random numbers in x
d1=: ?@#	x numbers from i.y with replacement
d2=: ?	x numbers from i.y without replacement
m3=: 9!:1@< .@ (+ / . ^ & 2@ (6!:0@]))	Randomizing random number seed
m4=: (> :@? % 2147483647&[) @ \$ & 2147483646	Random numbers between 0 and 1
m5=: { ~ ? ~@#	Random shuffle of y

11A. Inverse

J provides a comprehensive calculus of inverses:

a0=: I=: ^:_1	Inverse (adverb)
m1 =: ^ I	Natural log (^.); Inverse exponential
m2 =: ^.I	Exponential
m3 =: 10&^.	Base 10 log
m4 =: m3 I	Inverse base 10 log (10&^)
m5 =:] -: m4@m3	Tautology (test that m4 is left inverse)
m6 =:] -: m3@m4	Tautology (test that m4 is right inverse)
m7 =: ssc=: +/\	Sum scan (subtotals)
m8 =: ssc I	Inverse sum scan; first differences
m9 =: (] -:m8@m7) * . (] -:m7@m8)	Tautology
m10=: assc=: -/\	Alternating sum scan
m11=: assc I	
m12=:] -:	
m13=: * /\ I	e.g. * /\ m13 3 1 4 15 9 26 5 3
m14=: % /\ I	e.g. % /\ m14 3 1 4 15 9 26 5 3
m15=: ~: /\ I	e.g. ~: /\ m15 1 1 0 1 1 0 1 1
m16=: = /\ I	e.g. = /\ m16 1 1 0 1 1 0 1 1
m17=: + /\ . I	e.g. + /\ . m17 3 1 4 15 9 26 5 3
m18=: - /\ . I	e.g. - /\ . m18 3 1 4 15 9 26 5 3
m19=: * /\ . I	e.g. * /\ . m19 3 1 4 15 9 26 5 3
m20=: % /\ . I	e.g. % /\ . m20 3 1 4 15 9 26 5 3
m21=: ~: /\ . I	e.g. ~: /\ . m21 1 1 0 1 1 0 1 1
m22=: = /\ . I	e.g. = /\ . m22 1 1 0 1 1 0 1 1
d23=: # I	Expand; 'ab' -: 1 0 1 # 1 0 1 d23 'ab'

m24=: p: I	p (n) the number of primes less than n
m25=: x: I	Floating point approx. of a rational. e.g. m25 3r7
m26=: 1&+ I	Inverse increment; decrement
m27=: +&1 I	Inverse increment; decrement
m28=: >: I	Inverse increment; decrement
m29=: _1&+ I	Inverse decrement; increment
m30=: +&_1 I	Inverse decrement; increment
m31=: -&1 I	Inverse decrement; increment
m32=: <: I	Inverse decrement; increment
m33=: 2&* I	Inverse double; halve
m34=: *&2 I	Inverse double; halve
m35=: +: I	Inverse double; halve
m36=: 0.5&* I	Inverse halve; double
m37=: *&0.5 I	Inverse halve; double
m38=: %&2 I	Inverse halve; double
m39=: -: I	Inverse halve; double
m40=: ^&2 I	Inverse square
m41=: ^&3 I	Inverse cube
m42=: ^&0.5 I	Inverse square root
m43=: ^&1r3 I	Inverse cube root
m44=: 2&^ I	Inverse 2 with power; base 2 log
m45=: 10&^ I	Inverse 10 with power; base 10 log
m46=: 2&! I	Inverse triangular number. e.g. +/i.<.2&! I m
m47=: +~ I	Inverse double
m48=: *~ I	Inverse square
m49=: ^~ I	e.g. x^x=: ^~ I 12
m50=: (3&+)&(%&2)I -: (%&2 I)&(3&+ I)	Inverse of composition is composition of inverses

These inverses may be illustrated as follows:

```

x=: 2 3 5 7
,.(] ; m1 ; m2 ; m1@m2) x
+-----+
| 2 3 5 7 |
+-----+
| 0.6931472 1.09861 1.60944 1.94591 |
+-----+
| 7.38906 20.0855 148.413 1096.63 |
+-----+
| 2 3 5 7 |
+-----+

(] ; m7 ; m8 ; m9) x
+-----+
| 2 3 5 7 | 2 5 10 17 | 2 1 2 2 | 1 |
+-----+

(] ; m10 ; m11 ; m11@m10) x
+-----+
| 2 3 5 7 | 2 _1 4 _3 | 2 _1 2 _2 | 2 3 5 7 |
+-----+

```

11B. Duality

$h \vdash f \& g$ asserts that h is the dual of f under (or *with respect to*) g

m0 =: A=: +/ % #	Arithmetic mean
m1 =: H=: A &. (%"_)	Harmonic mean
m2 =: M=: A &. (^&p)	Generalized mean
a3 =: N=: (^&) (A&.)	L-x norm; 3-norm is 3 N y
m4 =: +/ &. (*:"_) @ +.	Magnitude e.g. m4 3j4
a5 =: each=: &.>	Each (f each applies f to each box)
m6 =: ^ .: -&.j.	Sine
m7 =: sin -: sinh&.j.	Tautology
m8 =: tan -: tanh&.j.	"
m9 =: sinh -: sin&.j.	"
m10=: cosh -: cos& j.	"
m11=: tanh -: tan&.j.	"
v12=: <. -: >.&.-	Tautologies
v13=: >. -: <.&.-	Tautologies
d14=: * -: +&.^.	Tautology
d15=: * -: +&.(10&^.)	Tautology
v16=: % -: -&.^.	Tautologies
d17=: + -: *&.^	Tautology
d18=: + -: *&.(10&^)	Tautology
v19=: - -: %&.^	Tautologies
v20=: % .: %&. :	Tautologies
v21=: % .: %&.(+@ :)	Tautologies
m22=: +/\ -: +/\.&. .	Tautology
m23=: +/\ .: +/\ &. .	Tautology
a24=: BW=: 1 : 'u./&.#:'	Bitwise adverb
m25=: *. BW	Bitwise AND. e.g. m25 _1 100 200
m26=: +. BW	Bitwise OR e.g. m26 100 200
m27=: ~: BW	Bitwise XOR e.g. m27 100 200

m28=: i.&.(p:^:_1)	The primes less than n
m29=: totient=: * -. %@ ~. &. q:	Euler's totient function
m30=: .&. ;:	Reverse the words; e.g. m30 'three score and ten years'
n31=: a=: 'abcdefghijklmnopqrstuvwxyz'	Space and alphabet
m32=: encrypt=: (#a)& @>: &.(a&i.)	Julius Caesar's cypher. e.g.
m33=: decrypt=: (#a)& @<: &.(a&i.)	decrypt encrypt x=: 'from sea to sea'
m34=: J=: 1& . &.#:	Survivor number in the Josephus problem of order n
m35=: ar=: >:@]	Increment right argument
m36=: dr=: <:@[Decrement right argument
m37=: dl=: <:@[Decrement left argument
m38=: test =: #.@(, &*)	Ackermann test
m35=: ack=: ar`ar`(dl ack 1:)`(dl ack[ack dr)@.test	Ackermann fn
m36=: 0&ack -: >:&.(3&+)	Ackermann 0
m37=: 1&ack -: 2&+&.(3&+)	Ackermann 1
m38=: 2&ack -: 2&*&.(3&+)	Ackermann 2
m39=: 3&ack -: 2&^&.(3&+)	Ackermann 3
m40=: 4&ack -: ^/@#&2&.(3&+)	Ackermann 4
m41=: 5&ack -: 3 : '^/@#&2^:(1+y.)&.(3&+) 1'	Ackermann 5

11C. Transformations

<code>m0=:] i."1&. : 1 _1"_</code>	Column node table from connection table columns (inverse to m3)
<code>m1=:] (i."1) 1 _1"_</code>	Node table from connection table rows (inverse to m4)
<code>m2=: [: -/] =/ [: i. [:>: [:>./ ,</code>	Row connection table from node table columns
<code>m3=: [: : m2</code>	Column connection table from node table columns (inverse to m0)
<code>m4=: [: -/"2] =/ [: i. [:>: [:>./ ,</code>	Row connection table from node table rows (inverse to m1)

```

      y
1  1  0  0  0
0 _1  0  1  1
_1  0  1 _1  0
0  0 _1  0 _1

```

```

      m0 y
0 0 2 1 1
2 1 3 2 3

```

```

      m3 m0 y
1  1  0  0  0
0 _1  0  1  1
_1  0  1 _1  0
0  0 _1  0 _1

```

12A. Finance

d0=: %@>:@(] % 100"_) #. .@[Present value flows x at y%
d1=: >:@(] % 100"_) #. [Future value of flows x at interest rate y
d2=: ([: >: [% 100"_) ^/]	Amount of 1 at x% for y periods
d3=: 13 : ' : y.%-.(>:y.) ^/ -x. '	Annuity coefficient: periods x at rate y
d4=: [d3 [: m5]	Annuity coefficient: periods x at y %
m5=: 0.01&*	Rate from percent
m6=: 100&*	Percent from rate
d7=: <: @ ((^%)~ >:)	Modal rate from annual rate
d8=: stretch=: [\$] , (\$, : @ { :)	Stretch y to length x
d9=: */\ @ (1&,) @ (stretch >:)	Accumulate at y for period x
m10=: (% { .) @ (.@(+/\)@i.@>:)	Outstanding balances on rule of 78
m11=: } . % } :	Rate of change from amounts
m12=: 2&((%~)/\)	Rate of change from amounts
d13=: 4 : 'r*+/\y.%r=:*/\1,x.\$~<:#y. '	Accumulate deposits y at x
d14=: [+2: -/\ [: >./\ 0: , [: +/\ -	Work done; x=capacity, y=demand

Convert an annual interest rate of 7% to a monthly rate.

```

]m=: 12 d7 0.07
0.00565415
(1 + m)^12
1.07

```

Calculate the amount of 1 for 5 periods at rates of 7% for the first two years and 6% thereafter. The function d9 uses d8 to fill out the term with the last given rate.

```

5 d9 0.07 0.07 0.06
1 1.07 1.1449 1.21359 1.28641 1.36359

```

Calculate the declining balances for a year for 1 using the rule of 78.

```

      ,.m10 12
      1
0.846154
0.705128
0.576923
0.461538
0.358974
0.269231
0.192308
0.128205
0.0769231
0.0384615
0.0128205
      0
      x: m10 12
1 11r13 55r78 15r26 6r13 14r39 7r26 5r26 5r39 1r13 1r26 1r78 0
      78 * m10 12
78 66 55 45 36 28 21 15 10 6 3 1 0

```

Calculate the balances after periodic payments of 100, 100, 200, and 200 are made at rates of 10%, 10%, and 5%.

```

1.10 1.10 1.05 m13 100 100 200 200
100 210 431 652.55

```

13A. Inside Boxes

a0=: each=: &.>	Apply verb to each box
m1=: open=: -:> :: 1:	Test if open (not boxed)
v2=: fmt=: ":	Format
m3=: just=: 3 : '9!:3 (2) [9!:17 y.'	Set function display to boxed form, and set justification within boxes to y
m4=: L. = 0:	Test if open
m5=: < -: {:@;~	Test if open
m6=: 32&>@(3!:0)	Test if open
m7=: <^:(L. = 0:)	Box if open

The adverb each applies its verb argument to the "inside" of each box of the argument of the resulting function, as illustrated below:

```
]c=: <"1 <"2 i.3 3 3 2
```

```
+-----+
|+-----+|+-----+|+-----+| | | | | | | | | | |
||0 1| 6 7|12 13||18 19|24 25|30 31||36 37|42 43|48 49||
||2 3| 8 9|14 15||20 21|26 27|32 33||38 39|44 45|50 51||
||4 5|10 11|16 17||22 23|28 29|34 35||40 41|46 47|52 53||
|+-----+|+-----+|+-----+|
+-----+
```

```
|. each c
```

```
+-----+
|+-----+|+-----+|+-----+| | | | | | | | | | |
||12 13| 6 7|0 1||30 31|24 25|18 19||48 49|42 43|36 37||
||14 15| 8 9|2 3||32 33|26 27|20 21||50 51|44 45|38 39||
||16 17|10 11|4 5||34 35|28 29|22 23||52 53|46 47|40 41||
|+-----+|+-----+|+-----+|
+-----+
```

```
|. each each c
```

```
+-----+
|+-----+|+-----+|+-----+| | | | | | | | | | |
||4 5|10 11|16 17||22 23|28 29|34 35||40 41|46 47|52 53||
||2 3| 8 9|14 15||20 21|26 27|32 33||38 39|44 45|50 51||
||0 1| 6 7|12 13||18 19|24 25|30 31||36 37|42 43|48 49||
|+-----+|+-----+|+-----+|
+-----+
```

```
|. each each each c
```

```
+-----+
|+-----+|+-----+|+-----+|
+-----+
```

```

|+---+|+---+|+---+|+---+|+---+|+---+|+---+|+---+|+---+| | | | | | | | | |
|0|1|6|7|12|13|18|19|24|25|30|31|36|37|42|43|48|49|
|+---+|+---+|+---+|+---+|+---+|+---+|+---+|+---+|+---+|
|2|3|8|9|14|15|20|21|26|27|32|33|38|39|44|45|50|51|
|+---+|+---+|+---+|+---+|+---+|+---+|+---+|+---+|+---+|
|4|5|10|11|16|17|22|23|28|29|34|35|40|41|46|47|52|53|
|+---+|+---+|+---+|+---+|+---+|+---+|+---+|+---+|+---+|
|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|
|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|

```

The display of a non-uniform boxed array illustrates the fact that boxed elements are normally justified to the top left. For example:

```

|a=: 2 2$('Baker';'John';('Apt 316','Temple St','York, Pa');617 204 4567
+-----+
|Baker      |John      |
+-----+
|Apt 316    |617 204 4567|
|Temple St  |            |
|York, Pa   |            |
+-----+

```

When applied to a boxed right argument, the left argument of the format function specifies the justification within a display, using 0, 1, and 2 for top, centre, bottom, and for left, centre, right:

```

fmt=: "
1 1 fmt a Centred rows and columns
+-----+
| Baker | John |
+-----+
|Apt 316 |      |
|Temple St|617 204 4567|
|York, Pa |      |
+-----+
; /all=: 3 3 #: i. 3 3
+-----+
|0 0|1 0|2 0|
|0 1|1 1|2 1|
|0 2|1 2|2 2|
+-----+

$ all fmt a
3 3 7 25

<"2 all fmt a
+-----+
|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+| | |
||Baker      |John      || Baker      | John      || Baker      | John      ||
|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|+-----+|
||Apt 316    |617 204 4567||Apt 316    |617 204 4567||Apt 316    |617 204 4567||

```

Temple St	York, Pa	Temple St	York, Pa	Temple St	York, Pa
Baker John		Baker John		Baker John	
Apt 316	Temple St 617 204 4567	Apt 316	Temple St 617 204 4567	Apt 316	Temple St 617 204 4567
Baker John		Baker John		Baker John	
Apt 316	Temple St	Apt 316	Temple St	Apt 316	Temple St
York, Pa	617 204 4567	York, Pa	617 204 4567	York, Pa	617 204 4567

The diagonal elements of this display are also given by the scalar left arguments 0, 1, and 2. Moreover, 9!:16 can be used to set the default justification of boxed displays, as it is in the function just, which first sets the function display to the boxed form. For example:

```
totient=: * -. @ % ~ . & . q:
just 1
```

totient

*	-. @ % ~ . & . q:
---	-------------------

just 2

totient

*	-. @ % ~ . & . q:
---	-------------------

+-----+

13B. Character

m0=: >:@(=&' ' ' ')#]	Double quotes in y (for execute)
m1=: \: ' ' &~:	Move all blanks to end of list
m2=: /: ' ' &=	Move all blanks to end of list
m3=: \: ' ' &=	Move all blanks to beginning of list
m4=: /: ' ' &~:	Move all blanks to beginning of list
m5=: [: -. 1: #.~ 1: , ' ' " _ =]	Negative of count of trailing blanks in y
m6=: (m5 .])"1	Justify y right
m7=: ' ' &=	Locate blanks in y
d8=: [: +/ [<: /~ ([:m7])# [:i . [:#]	Word index in y from character index x
m9=: -@< .@- :@(+/)@(* . / \ .)@(' ' &=)	Rotation for centering left justified text
m10=: m9"1 . "_1]	Center left justified text
m11=:] .~ - @(+/)@(* . / \ .)@(' ' &=)	Justify right
m12=:] .~ (+/)@(* . / \ .)@(' ' &=)	Justify left
m13=: +/\@}:@(- //. @ (' () ' " _ = /]))	Depth of parentheses in y
d14=: < .@-:@([- #@]) . - @[{ .]	Center y in field x wide

13C. Type Change

m0=: [: 0&": 10"_ #. #:	Character binary format of integer y
m1=: 1: ": :@(10 10"_ #: >:@i.)	Character form of column # heading
m2=:] { '0123456789abcdef' "_	Character form of #, bases 2 thru 16
m3=: <"1	Box lists of array y
m4=: 16"_#.'0123456789abcdef' "_ i.]	Decimal from hexadecimal characters y
m5=: 10"_ #. '0123456789' "_ i.]	Decimal from decimal characters y
m6=: ". .	Convert rows of y to numbers (default 0)
d7=: []`[@.(0: = #@)] ".@]	# from character y, x if y not a number
m8=: }.@(" .@('0 '&,@(*./@(e.&'0123456789'@)) #])))	# from character, empty if not character
m9=: ". .	Execute each row of character table y
m10=: +/@: ". .	Sum of numbers in character table y
m11=: ":@, .	Format numeric list y as table
d12=: [:}. "1[:": (10"^([])+([: ,.])	Format list y as x-wide col with leading 0s
d13=: [n24 } m25@]	Format y with 0s replaced by x
d14=: ": "_1	Row-wise format of y by x
d15=: ": :	Fmt y in fields <.x wide and 1 x decimals
m16=: #@":	Number of characters in format of y
n17=: rn=: 'MDCLXVI'	Roman numerals
m18=: rn&i.	Roman numerals y in code
m19=: {&1000 500 100 50 10 5 1	Decode decimal values from indices y
m20=: [: [: - /]+//.~0: ,~2:</\]	Decimal number from values of Roman
m21=: DFR=: m20@m19@m18	Decimal from Roman
m22=: RFD=: #&rn@(5 2 5 2 5 2 5&#:))	Roman from Decimal (no refinements)
v23=: [: <"1 \$ #: [: m26 [: , '1' "_ =]	Indices of '1' in character table y
n24=: [`(v23@{:@])`({.@])	Gerund for amend

m25=: [: ":] ,:] = 0:	Laminate boolean array locating free 0s in y
m26=: +/ { . \:	Indices of 1s in Boolean list y

```

      y
12.34 4 0    7.8
      0 6 0    8.9
123.45 0 0 10.11
      ' ' d13 y
12.34 4      7.8
      6      8.9
123.45      10.11

```

```

      DFR 'MCMIX'
1909
      RFD 1909
MDCCCCVIIII

```

14A. Execution Time and Space

v0=: time=: 6!:2	Time to execute y (or avg of x executes)
m1=: space=: 7!:2	Space used to execute y
v2=: ts=: 6!:2, 7!:2@]	Time and space

The foregoing phrases may be used to obtain the time and space required to execute a sentence. For example:

```

r=: report=: ?3 4 5$10
r;(/:~r);(/:"1~r)
+-----+
|6 6 9 3 5|6 4 7 9 7|3 5 6 6 9|
|8 0 0 5 6|2 0 7 3 6|0 0 5 6 8|
|0 3 0 4 6|7 9 3 2 9|0 0 3 4 6|
|5 9 8 5 0|7 7 6 0 6|0 5 5 8 9|
|
|6 4 7 9 7|6 6 9 3 5|4 6 7 7 9|
|2 0 7 3 6|8 0 0 5 6|0 2 3 6 7|
|7 9 3 2 9|0 3 0 4 6|2 3 7 9 9|
|7 7 6 0 6|5 9 8 5 0|0 6 6 7 7|
|
|8 2 4 7 4|8 2 4 7 4|2 4 4 7 8|
|2 2 3 1 4|2 2 3 1 4|1 2 2 3 4|
|8 9 0 9 5|8 9 0 9 5|0 5 8 9 9|
|5 3 9 4 2|5 3 9 4 2|2 3 4 5 9|
+-----+

time '/:~r'
0

100 time '/:~r'
0.0005

100 time '/: "1~r'
0.0005

space '/: "1~r'
4160

```

100 ts ' /: "1~r'
0.0005 4160

14B. Date & Time

m0=: 6!:0	System time stamp yyyy mm dd hh mm ss (argument y needed but ignored)
m1=: [:<._3:{.]	hh mm ss part of time stamp
m2=: 1000"_ #.]	Base-1000 form of y
m3=: (':'_"_) _6 _3} ":	Format with ':' between fields
m4=: [:<._3:{.]	yyyy mm dd part of time stamp
m5=: ('/'_"_) 4 7} ":	Format with '/' between fields
m6=: [: m3 [: m2 [: m1 m0	Formatted time stamp hh:mm:ss
m7=: [: m5 [: m2 [: m4 m0	Formatted date stamp yyyy/mm/dd
m8=: (':'_"_) 2 5} [: ":	Formatted time stamp hh:mm:ss
1000"_ #. _3: {. [: <. m0	
m9=: ('/'_"_) 4 7} [: ":	Formatted date stamp yyyy.mm/dd
1000"_ #. 3: {. [: <. m0	
m10=: 100"_#._100"_ 3:{.	yymmdd from cyy mm dd (y neglected)
m0	
m11=: 0: ~:/ . = 4 100	Is y a leap year?
400"_ /]	
m12=: 28"_ + m11@]	Number of days in February of year y
d13=: 31"_ - 2: 7: [31 - 2 7 x: days in month x, not = 1
d14=: d13`m12@.([=1:)	Number of days in month x of year y
m15=: ('0123456789 ' "_	
i.[:":[: . 3: {. 6!:0) {	Current date in dd.mm.yy fmt, neglect y
('0123456789.' "_)	
m16=: ((12"_ <: {.),2:) {	am or pm depending on first atom of y
'apm' "_	
m17=: ([: m3 m2) , (''_"_	Formatted 3-atom time y in 'm' form
, m16)	
m18=:	
[[:":[:(1: .)]100"_ [:m4	Formatted date in mm dd yy form
m0	
m19=: '/'_"_([#[:i.#)@('	Replace blanks in y by '/'
' "_=])}]	

m20=: m19@m18	Formatted date in mm/dd/yy form
m21=: >:@(365&* + m22)@(-&1601)	# of New Year's Day, Gregorian year y; m21 1601 is 1
m22=: -/@:<.@(%&4 100 400)"0	# of leap days in y years (Clavian corr.)
m23=: 7& @m21	Day of week year y begins (0=Sunday)

Number of New Year's Day for Gregorian years.

```

    m21 1601 1602 2001
1 366 146098

```

Number of leap days in y years

```

    m22 0 1 4 100 400 2000
0 0 1 24 97 485

```

Day of week year y begins (0 is Sunday)

```

    m23 1900 2000
1 6

```

15A. Case Statements

If `ag` generates indices for a gerund `ger`, we will call `ag` an *agenda* function, and the function `ger@.ag` a *case statement*. For example:

```

ger=: f`g`h
ag=: #@$
case=: ger@.ag
f=: -:
g=: +:
h=: *:
(case 3);(case 2 3 4 5);(case i.3 4)
+-----+
|1.5|4 6 8 10| 0 1 4 9|
|   |         |16 25 36 49|
|   |         |64 81 100 121|
+-----+

f=: *:
(case 3);(case 2 3 4 5);(case i.3 4)
+-----+
|9|4 6 8 10| 0 1 4 9|
| |         |16 25 36 49|
| |         |64 81 100 121|
+-----+

```

Certain agendas prove to be useful with a variety of gerunds: for example, the *rank* used above, as well as various classifications such as negative, zero, and positive; integral or fractional; real or complex; numeric or character; boxed or open; and the *depth* of boxing. Since indices may be negative, the result of an agenda may be negative; thus the case `f`g`h@.*` applies `f` if the argument is zero, `g` if it is positive, and `h` if it is negative.

Since an agenda such as the hook `=<.` (which tests for fractional or integral) might invoke a domain error (when the argument is character or boxed), it is often useful to extend an agenda to produce a result in such a case. If this result is `_1`, the corresponding function in the gerund (perhaps `h=: 'ERROR' "_"`) may be simply appended to the normal cases. Thus, the test for integral may be defined (using *adverse*) as `(=<.) :: _1`: For example:

```

F=: (=<.) :: _1:
F"0 x=: 0.5 _2 2
0 1 1

```

```

F 'abcd'
_1

```

m0=: ~. ,. #/.~	Nub and count
m1=: ({.,#)/.~	Nub and count
a2=: et=: :: _1:	Error in tail position
m3=: I=: (-:<.)et	Integral
m4=: C=: -.@(-:+) et	Complex
a5=: ep=: :: _2:	Error in penultimate position
m6=: S=: *ep	Signum test with error in penultimate
m7=: B=: -.@(-:>) :: 0:	Boxed
m8=: R=: #@	Rank

An "or" over an agenda (that is, applying a given function for any one of several cases distinguished by the agenda) can be achieved by placing the same function in several places in the gerund. Moreover, agendas may be used in combination, as illustrated below:

```

c=: co`cb@.B Executes co if open; cb if boxed
co=: -@|`+@.C Minus magnitude if real; conjugate
cb=: ]`(|.&.>)`(|:&.>)@.(R@>) Reverse if opened is list;
transpose

```

```

c <i. 2 4 if opened is a table
cb <i.2 4 The case chosen by the agenda B

```

```

+----+
|0 4|
|1 5|
|2 6|
|3 7|
+----+

```

```

c 3j4 5 6j7 (Open) complex argument
3j_4 5 6j_7 Conjugate

```

```

co 3j4 5 6j7 The case chosen by the agenda B

```

3j_4 5 6j_7

The complete definition of c can be seen by *fixing* it:

```

c f.
-@|`+@.(-.@(-: +) ::(_1:))`([`(|.&.>)`(|:&.>)@.(@$@>))@.(-.@(-:
>) ::0:)
```

15B. Miscellaneous

v0=. '.'&\$: : (2&{.@(;< ; ._1)@,)	Split y on x (on period for monad)
m1=: (i.@# = i.~) #]	Nub ~.
n2=: a.{~65 97+/i.26	A-Z, a-z

For example:

```

v0 'filename.ext'
+-----+
|filename|ext|
+-----+

';' v0 'filename;ext'
+-----+
|filename|ext|
+-----+

```

The following illustrates miscellaneous matters, including multiple assignments:

n=. 'psmith';(i.3 4);100 200	assign local to explicit definition
n=: 'psmith';(i.3 4);100 200	assign global
'one two three'=: n	assign names
'a b c'=: i.3 4	assign rows of matrix
'a b c d'=: :i.3 4	assign cols of matrix
'a b c'=: 1 2 3;'two';3 3\$3	assign opened elements
[x=: i.5	assign and display result
v=: +:@*:	assign name to verb
([v) i.5	invoke verb, return argument
(;v) i.5	invoke verb, return argument linked to result

References

1. Abramowitz, Milton, and Irene A. Stegun, *Handbook of Mathematical Functions*, National Bureau of Standards, 1964
2. de Kerf, Joseph, *The Common Mean and APL*, Vector 11 3, Jan.1995
3. Graham, Ronald L., Donald E. Knuth, and Oren Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Addison-Wesley, 1988
4. Hui, Roger K.W., *Some Uses of { and }*, APL87, APL Quote-Quad, Volume 17, Number 4, 1987
5. Hui, Roger K.W., K.E. Iverson, Iverson, K.E., *Calculus*, Iverson Software Inc., 1994

Index

A

- a. [C. Specimens p3](#) · [A. Bond or Curry p5](#) · [B. Merge & Amend p14](#) · [B. Locating & Selecting p16](#) · [B. Miscellaneous p54](#)
- A. [A. Conventions p1](#) · [A. Bond or Curry p5](#) · [A. Indexing p13](#) · [A. Intervals p15](#) · [A. Structural p18](#) · [A. Sorting p21](#) · [A. Permutations p24](#) · [C. Parity & Symmetry p26](#) · [A. Numbers & Counting p27](#) · [A. Matrix Algebra p32](#) · [A. Sums & Means p38](#) · [A. Inverse p44](#) · [A. Finance p47](#) · [A. Inside Boxes p48](#) · [A. Execution Time & Space p51](#) · [A. Case Statements p53](#) · [REFERENCES p55](#)
- a: [A. Indexing p13](#)
- abbreviated [A. Indexing p13](#) · [A. Permutations p24](#)
- Abramowitz [REFERENCES p55](#)
- abs [C. Ambivalent Functions p7](#)
- abstract [A. Intervals p15](#)
- accumulate(s) [A. Finance p47](#)
- accuracy [A. Numbers & Counting p27](#)
- accurate [A. Numbers & Counting p27](#)
- ackermann [B. Duality p45](#)
- acosh [A. Bond or Curry p5](#)
- adverse [A. Case Statements p53](#)
- agenda(s) [A. Case Statements p53](#)

algebra	A. Matrix Algebra p32 · C. Polynomials & Rational Functions p34
alphabet	A. Bond or Curry p5 · A. Sorting p21 · B. Duality p45
alphabetic	C. Specimens p3
ambivalent	A. Conventions p1 · C. Ambivalent Functions p7 · H. Gerunds p12
amend(s)	A. Indexing p13 · B. Merge & Amend p14 · C. Type Change p50
amended	B. Merge & Amend p14
amendment(s)	B. Merge & Amend p14
antilog(s)	A. Bond or Curry p5 · D. Transcendental Functions p35
APL	F. Hooks p10 · A. Indexing p13 · C. Special Matrices & Lists p20 · REFERENCES p55
approx	A. Numbers & Counting p27 · E. Approximation p42 · A. Inverse p44
approximated	E. Quadrature & Simpson's Rule p36
approximation(s)	F. Hooks p10 · A. Numbers & Counting p27 · E. Quadrature & Simpson's Rule p36 · E. Approximation p42
arccos	A. Bond or Curry p5
arccosh	A. Bond or Curry p5
arcsin	A. Bond or Curry p5 · C. Parity & Symmetry p26
arcsine	C. Parity & Symmetry p26
arcsinh	A. Bond or Curry p5
arctan	A. Bond or Curry p5
arctanh	A. Bond or Curry p5
arithmetic	A. Conventions p1 · A. Bond or Curry p5 · D. Arithmetic p30 · A. Sums & Means p38 · B. Duality p45
ascii	A. Bond or Curry p5
asin	A. Bond or Curry p5

<code>asinh</code>	A. Bond or Curry p5
<code>assert(s)</code>	A. Conventions p1 · B. Duality p45
<code>assignment(s)</code>	A. Conventions p1 · B. Ranking & Classification p22 · C. Grading p23 · B. Miscellaneous p54
<code>atan</code>	A. Bond or Curry p5
<code>atanh</code>	A. Bond or Curry p5
<code>atom(s)</code>	C. Specimens p3 · A. Intervals p15 · B. Locating & Selecting p16 · C. Test p17 · B. Ranking & Classification p22 · B. Grids p28 · B. Date & Time p52
<code>atop</code>	C. Polynomials & Rational Functions p34
<code>axe(s)</code>	A. Indexing p13 · C. Parity & Symmetry p26

B

<code>b.</code>	D. Adverbs & Conjunctions p8 · A. Intervals p15
<code>barchart(s)</code>	D. Plotting p41
<code>binary</code>	C. Representations p29 · C. Math & Stats p40 · C. Type Change p50
<code>binomial(s)</code>	D. Function Tables p4 · A. Bond or Curry p5 · H. Gerunds p12 · A. Numbers & Counting p27 · C. Polynomials & Rational Functions p34
<code>bits</code>	A. Bond or Curry p5
<code>bitwise</code>	B. Duality p45
<code>boolean(s)</code>	C. Specimens p3 · D. Adverbs & Conjunctions p8 · F. Hooks p10 · B. Merge & Amend p14 · A. Intervals p15 · B. Locating & Selecting p16 · C. Test p17 · B. Partition p19 · A. Sorting p21 · A. Numbers & Counting p27 · B. Grids p28 · C. Representations p29 · D. Plotting p41 · C. Type Change p50

box [A. Conventions p1](#) · [C. Specimens p3](#) · [D. Adverbs & Conjunctions p8](#) · [B. Partition p19](#) · [C. Special Matrices & Lists p20](#) · [A. Permutations p24](#) · [B. Grids p28](#) · [F. Geometry p37](#) · [B. Duality p45](#) · [A. Inside Boxes p48](#) · [C. Type Change p50](#)

boxed [C. Specimens p3](#) · [E. Explicit Definitions p9](#) · [A. Indexing p13](#) · [B. Locating & Selecting p16](#) · [C. Special Matrices & Lists p20](#) · [A. Sorting p21](#) · [A. Permutations p24](#) · [A. Numbers & Counting p27](#) · [F. Geometry p37](#) · [A. Inside Boxes p48](#) · [A. Case Statements p53](#)

boxes [A. Inside Boxes p48](#)

C

C. [A. Conventions p1](#) · [B. Loading Scripts p2](#) · [C. Specimens p3](#) · [A. Bond or Curry p5](#) · [C. Ambivalent Functions p7](#) · [C. Test p17](#) · [C. Special Matrices & Lists p20](#) · [C. Grading p23](#) · [A. Permutations p24](#) · [C. Parity & Symmetry p26](#) · [C. Representations p29](#) · [C. Polynomials & Rational Functions p34](#) · [C. Math & Stats p40](#) · [C. Transformations p46](#) · [C. Type Change p50](#)

calculus [C. Polynomials & Rational Functions p34](#) · [E. Quadrature & Simpson's Rule p36](#) · [A. Inverse p44](#) · [REFERENCES p55](#)

catenation [B. Merge & Amend p14](#)

circle [A. Bond or Curry p5](#) · [D. Adverbs & Conjunctions p8](#) · [E. Complex Numbers p31](#) · [F. Geometry p37](#)

circular [D. Adverbs & Conjunctions p8](#)

circumference [F. Geometry p37](#)

coeff(s) [A. Matrix Algebra p32](#) · [C. Polynomials & Rational Functions p34](#) · [A. Sums & Means p38](#)

coefficient(s)	D. Function Tables p4 • A. Bond or Curry p5 • D. Adverbs & Conjunctions p8 • H. Gerunds p12 • A. Numbers & Counting p27 • C. Polynomials & Rational Functions p34 • E. Quadrature & Simpson's Rule p36 • A. Finance p47
coeffieicnts	C. Polynomials & Rational Functions p34
compared	E. Complex Numbers p31
compares	F. Hooks p10 • C. Parity & Symmetry p26
comparisons	A. Sorting p21
component(s)	A. Conventions p1 • C. Ambivalent Functions p7
composition(s)	C. Polynomials & Rational Functions p34 • A. Inverse p44
conjugate(s)	E. Complex Numbers p31 • A. Case Statements p53
conjunction(s)	A. Conventions p1 • A. Bond or Curry p5 • D. Adverbs & Conjunctions p8 • E. Explicit Definitions p9 • B. Partition p19 • C. Parity & Symmetry p26 • C. Polynomials & Rational Functions p34
constants	A. Numbers & Counting p27

D

D.	D. Function Tables p4 • D. Adverbs & Conjunctions p8 • C. Special Matrices & Lists p20 • D. Arithmetic p30 • D. Transcendental Functions p35 • D. Plotting p41
date(s)	B. Date & Time p52
decimals	C. Type Change p50
decode(s)	C. Type Change p50
derivative(s)	C. Parity & Symmetry p26 • C. Polynomials & Rational Functions p34 • E. Quadrature & Simpson's Rule p36
derived	C. Grading p23 • C. Parity & Symmetry p26

determinant(s)	C. Parity & Symmetry p26 · A. Matrix Algebra p32 · F. Geometry p37 · C. Math & Stats p40
diagonal(s)	B. Merge & Amend p14 · C. Special Matrices & Lists p20 · B. Rotations & Reflections p25 · A. Inside Boxes p48
dimension(s)	B. Locating & Selecting p16 · A. Structural p18 · C. Parity & Symmetry p26 · B. Linear Vector Functions p33
distribution(s)	B. Locating & Selecting p16 · C. Math & Stats p40
divided	D. Adverbs & Conjunctions p8
dividing	D. Arithmetic p30 · E. Quadrature & Simpson's Rule p36
divisible	B. Locating & Selecting p16
divisor(s)	A. Numbers & Counting p27 · D. Arithmetic p30
dot(s)	C. Math & Stats p40
doubles	A. Numbers & Counting p27
dual(s)	B. Duality p45
duality	B. Duality p45
dyad(s)	A. Conventions p1 · C. Ambivalent Functions p7 · B. Grids p28
dyadic	A. Conventions p1 · C. Ambivalent Functions p7

E

e.	A. Bond or Curry p5 · A. Intervals p15 · B. Locating & Selecting p16 · C. Test p17 · C. Special Matrices & Lists p20 · A. Numbers & Counting p27 · C. Type Change p50
E.	A. Bond or Curry p5 · E. Explicit Definitions p9 · G. Verb Trains p11 · B. Locating & Selecting p16 · C. Special Matrices & Lists p20 · E. Complex Numbers p31 · E. Quadrature & Simpson's Rule p36 · E. Approximation p42 · REFERENCES p55
enclosing	A. Conventions p1 · E. Explicit Definitions p9

encrypt	B. Duality p45
equals	C. Special Matrices & Lists p20 · C. Parity & Symmetry p26 · B. Linear Vector Functions p33
equivalence	A. Conventions p1
euler(s)	A. Numbers & Counting p27 · B. Duality p45
evaluate(s)	C. Special Matrices & Lists p20
evaluated	C. Parity & Symmetry p26
executable(s)	A. Conventions p1
executes	A. Execution Time & Space p51 · A. Case Statements p53
executing	C. Parity & Symmetry p26
expand(s)	C. Special Matrices & Lists p20 · C. Polynomials & Rational Functions p34 · A. Inverse p44
expanding	B. Locating & Selecting p16
exponent(s)	A. Numbers & Counting p27 · C. Polynomials & Rational Functions p34
exponential(s)	A. Bond or Curry p5 · C. Parity & Symmetry p26 · A. Numbers & Counting p27 · D. Transcendental Functions p35 · E. Approximation p42 · A. Inverse p44

F

f.	A. Conventions p1 · D. Adverbs & Conjunctions p8 · B. Linear Vector Functions p33 · F. Geometry p37 · A. Case Statements p53
factor(s)	A. Numbers & Counting p27
factorial(s)	D. Adverbs & Conjunctions p8 · A. Numbers & Counting p27 · C. Polynomials & Rational Functions p34
factorization(s)	A. Numbers & Counting p27
fork(s)	C. Parity & Symmetry p26

[fraction\(s\)](#) [A. Conventions p1](#) · [C. Specimens p3](#) · [F. Hooks p10](#) · [A. Numbers & Counting p27](#) · [C. Representations p29](#)

[fractional](#) [C. Specimens p3](#) · [C. Representations p29](#) · [A. Case Statements p53](#)

[fuzzed](#) [A. Intervals p15](#)

G

[game\(s\)](#) [B. Ranking & Classification p22](#)

[gamma](#) [D. Transcendental Functions p35](#) · [C. Math & Stats p40](#)

[geometric](#) [A. Sums & Means p38](#)

[geometry](#) [F. Geometry p37](#)

[gerund\(s\)](#) [H. Gerunds p12](#) · [C. Type Change p50](#) · [A. Case Statements p53](#)

[gerundial](#) [H. Gerunds p12](#)

[grade\(s\)](#) [A. Sorting p21](#) · [C. Grading p23](#) · [C. Parity & Symmetry p26](#)

[grading](#) [C. Grading p23](#)

H

[H.](#) [H. Gerunds p12](#)

[halve\(s\)](#) [A. Bond or Curry p5](#) · [A. Inverse p44](#)

[handbook](#) [REFERENCES p55](#)

[head\(s\)](#) [A. Bond or Curry p5](#)

[hexadecimal](#) [C. Type Change p50](#)

[hilbert](#) [C. Special Matrices & Lists p20](#)

[hyperbolic\(s\)](#) [C. Parity & Symmetry p26](#)

I

i.	A. Conventions p1 · B. Loading Scripts p2 · C. Specimens p3 · D. Function Tables p4 · A. Bond or Curry p5 · D. Adverbs & Conjunctions p8 · E. Explicit Definitions p9 · F. Hooks p10 · G. Verb Trains p11 · H. Gerunds p12 · A. Indexing p13 · B. Merge & Amend p14 · B. Locating & Selecting p16 · C. Special Matrices & Lists p20 · B. Ranking & Classification p22 · C. Grading p23 · A. Permutations p24 · C. Parity & Symmetry p26 · A. Numbers & Counting p27 · B. Grids p28 · C. Representations p29 · D. Arithmetic p30 · E. Complex Numbers p31 · A. Matrix Algebra p32 · B. Linear Vector Functions p33 · C. Polynomials & Rational Functions p34 · E. Quadrature & Simpson's Rule p36 · F. Geometry p37 · A. Sums & Means p38 · C. Math & Stats p40 · D. Plotting p41 · A. Inverse p44 · B. Duality p45 · C. Transformations p46 · A. Finance p47 · A. Inside Boxes p48 · C. Type Change p50 · B. Date & Time p52 · A. Case Statements p53 · B. Miscellaneous p54
i:	G. Verb Trains p11
imaginary	C. Special Matrices & Lists p20 · A. Numbers & Counting p27 · E. Complex Numbers p31
increment(s)	A. Bond or Curry p5 · A. Inverse p44 · B. Duality p45
indexed	B. Locating & Selecting p16
indexing	A. Indexing p13
infinity	C. Representations p29
infix	B. Locating & Selecting p16
infixes	B. Locating & Selecting p16
integral(s)	C. Polynomials & Rational Functions p34 · E. Quadrature & Simpson's Rule p36 · A. Case Statements p53
integrating	E. Quadrature & Simpson's Rule p36

integration [C. Polynomials & Rational Functions p34](#)
inverses [A. Inverse p44](#)
invoke(s) [F. Hooks p10](#) · [A. Case Statements p53](#) · [B. Miscellaneous p54](#)
invoked [A. Conventions p1](#)

J

j. [C. Specimens p3](#) · [A. Bond or Curry p5](#) · [C. Special Matrices & Lists p20](#) · [E. Complex Numbers p31](#) · [D. Transcendental Functions p35](#) · [B. Duality p45](#)
jsoftware [A. Inside Boxes p48](#) · [REFERENCES p55](#)
languages [A. Indexing p13](#)
largest [B. Grids p28](#)
legendre [A. Numbers & Counting p27](#)
letter(s) [B. Loading Scripts p2](#)
limit(s) [D. Adverbs & Conjunctions p8](#) · [A. Intervals p15](#) · [D. Arithmetic p30](#)
linearity [B. Linear Vector Functions p33](#)
locale(s) [B. Loading Scripts p2](#) · [C. Specimens p3](#)
log(s) [A. Bond or Curry p5](#) · [C. Ambivalent Functions p7](#) · [E. Explicit Definitions p9](#) · [A. Numbers & Counting p27](#) · [D. Transcendental Functions p35](#) · [A. Inverse p44](#)

M

magnitude(s) [A. Numbers & Counting p27](#) · [E. Complex Numbers p31](#) · [A. Sums & Means p38](#) · [B. Duality p45](#) · [A. Case Statements p53](#)
math [D. Adverbs & Conjunctions p8](#) · [C. Math & Stats p40](#)

mathematical	A. Conventions p1 • A. Intervals p15 • REFERENCES p55
mathematics	G. Verb Trains p11 • REFERENCES p55
matrices	A. Conventions p1 • C. Specimens p3 • C. Special Matrices & Lists p20
matrix	A. Conventions p1 • A. Bond or Curry p5 • B. Rank p6 • E. Explicit Definitions p9 • B. Locating & Selecting p16 • C. Special Matrices & Lists p20 • A. Permutations p24 • C. Parity & Symmetry p26 • A. Matrix Algebra p32 • B. Linear Vector Functions p33 • F. Geometry p37 • C. Math & Stats p40 • B. Miscellaneous p54
membership(s)	B. Locating & Selecting p16
mnemonic(s)	A. Conventions p1
modal	A. Finance p47
multinomial	C. Polynomials & Rational Functions p34
multiplied	C. Polynomials & Rational Functions p34
multiplier(s)	A. Numbers & Counting p27
multiply	A. Bond or Curry p5

N

NB.	A. Conventions p1
neg	H. Gerunds p12
negate(s)	A. Bond or Curry p5 • C. Parity & Symmetry p26
negation	C. Parity & Symmetry p26
negint	A. Numbers & Counting p27
nub	B. Partition p19 • A. Numbers & Counting p27 • A. Case Statements p53 • B. Miscellaneous p54
numerical	C. Test p17

O

- o. [B. Loading Scripts p2](#) · [A. Bond or Curry p5](#) · [D. Adverbs & Conjunctions p8](#) · [C. Parity & Symmetry p26](#) · [A. Numbers & Counting p27](#) · [D. Transcendental Functions p35](#) · [F. Geometry p37](#)
- operator(s) [A. Sorting p21](#)

P

- p. [D. Adverbs & Conjunctions p8](#) · [E. Explicit Definitions p9](#) · [A. Numbers & Counting p27](#) · [C. Polynomials & Rational Functions p34](#) · [E. Quadrature & Simpson's Rule p36](#) · [A. Sums & Means p38](#)
- p: [A. Numbers & Counting p27](#) · [C. Math & Stats p40](#) · [A. Inverse p44](#) · [B. Duality p45](#)
- parentheses [B. Character p49](#)
- partition(s) [B. Partition p19](#) · [A. Numbers & Counting p27](#)
- partitioned [B. Partition p19](#)
- partitioning [B. Partition p19](#)
- plotting [D. Plotting p41](#)
- poisson [C. Math & Stats p40](#)
- polynomial(s) [D. Adverbs & Conjunctions p8](#) · [C. Polynomials & Rational Functions p34](#) · [E. Quadrature & Simpson's Rule p36](#) · [A. Sums & Means p38](#) · [C. Math & Stats p40](#)

power(s)	A. Conventions p1 · D. Adverbs & Conjunctions p8 · H. Gerunds p12 · A. Permutations p24 · A. Numbers & Counting p27 · C. Polynomials & Rational Functions p34 · D. Transcendental Functions p35 · A. Sums & Means p38 · A. Inverse p44
precedence(s)	C. Parity & Symmetry p26
prime(s)	B. Locating & Selecting p16 · C. Special Matrices & Lists p20 · A. Numbers & Counting p27 · C. Math & Stats p40 · A. Inverse p44 · B. Duality p45
primitive(s)	A. Conventions p1 · A. Bond or Curry p5 · C. Grading p23
probability	A. Bond or Curry p5

Q

q:	A. Numbers & Counting p27 · A. Inside Boxes p48
quadratic	A. Numbers & Counting p27 · C. Polynomials & Rational Functions p34

R

r.	A. Bond or Curry p5
radian(s)	A. Bond or Curry p5 · D. Adverbs & Conjunctions p8 · C. Parity & Symmetry p26 · A. Numbers & Counting p27
radius	F. Geometry p37
radix	A. Numbers & Counting p27
random	C. Specimens p3 · A. Sorting p21 · A. Permutations p24 · F. Geometry p37 · F. Random Numbers p43
randomizing	F. Random Numbers p43

rank(s)	A. Conventions p1 • C. Specimens p3 • B. Rank p6 • E. Explicit Definitions p9 • A. Indexing p13 • A. Structural p18 • C. Special Matrices & Lists p20 • A. Sorting p21 • B. Ranking & Classification p22 • A. Numbers & Counting p27 • B. Grids p28 • A. Case Statements p53
rational(s)	D. Adverbs & Conjunctions p8 • A. Numbers & Counting p27 • C. Polynomials & Rational Functions p34 • A. Inverse p44
raveled	A. Sorting p21
ravelled	B. Merge & Amend p14
reciprocal(s)	A. Bond or Curry p5 • A. Numbers & Counting p27 • C. Math & Stats p40
recursion	H. Gerunds p12
recursive	A. Bond or Curry p5 • A. Permutations p24
recursively	H. Gerunds p12
relational(s)	A. Intervals p15 • B. Locating & Selecting p16
rotated	E. Complex Numbers p31
rotates	B. Rotations & Reflections p25
rotation(s)	B. Rotations & Reflections p25 • F. Geometry p37 • B. Character p49

S

school(s)	C. Polynomials & Rational Functions p34
script(s)	B. Loading Scripts p2
self	C. Ambivalent Functions p7 • C. Special Matrices & Lists p20 • C. Polynomials & Rational Functions p34
signum	A. Intervals p15 • A. Case Statements p53
simplex	F. Geometry p37

sinh	A. Bond or Curry p5 · C. Parity & Symmetry p26 · D. Transcendental Functions p35 · B. Duality p45
sorting	A. Sorting p21 · C. Grading p23
square(s)	A. Conventions p1 · A. Bond or Curry p5 · D. Adverbs & Conjunctions p8 · G. Verb Trains p11 · A. Numbers & Counting p27 · C. Math & Stats p40 · E. Approximation p42 · A. Inverse p44
squared	D. Adverbs & Conjunctions p8

T

t.	C. Parity & Symmetry p26 · C. Polynomials & Rational Functions p34
tacit	C. Ambivalent Functions p7 · E. Explicit Definitions p9
tangent	D. Adverbs & Conjunctions p8
tanh	A. Bond or Curry p5 · D. Transcendental Functions p35 · B. Duality p45
tautologies	B. Duality p45
tautology	A. Conventions p1 · G. Verb Trains p11 · A. Intervals p15 · C. Polynomials & Rational Functions p34 · D. Transcendental Functions p35 · A. Sums & Means p38 · A. Inverse p44 · B. Duality p45
taylor	C. Polynomials & Rational Functions p34
technique(s)	B. Locating & Selecting p16
tensor	C. Parity & Symmetry p26
tie(s)	B. Ranking & Classification p22
train(s)	G. Verb Trains p11
transcendental	D. Transcendental Functions p35
transform(s)	C. Polynomials & Rational Functions p34

transformation(s) [C. Polynomials & Rational Functions p34](#) • [C. Transformations p46](#)

transitive [C. Math & Stats p40](#)

translation(s) [E. Explicit Definitions p9](#)

transpose(s) [C. Special Matrices & Lists p20](#) • [C. Parity & Symmetry p26](#) • [A. Case Statements p53](#)

transposed [E. Quadrature & Simpson's Rule p36](#)

transposition(s) [C. Parity & Symmetry p26](#)

U

u. [B. Duality p45](#)

V

verb(s) [A. Conventions p1](#) • [A. Bond or Curry p5](#) • [D. Adverbs & Conjunctions p8](#) • [G. Verb Trains p11](#) • [A. Intervals p15](#) • [B. Locating & Selecting p16](#) • [A. Structural p18](#) • [A. Sorting p21](#) • [A. Inside Boxes p48](#) • [B. Miscellaneous p54](#)

W

weight(s) [E. Quadrature & Simpson's Rule p36](#)

weighted [C. Polynomials & Rational Functions p34](#) • [E. Quadrature & Simpson's Rule p36](#) • [B. E Pluribus Unum p39](#) • [C. Math & Stats p40](#)

X

- x. [B. Loading Scripts p2](#) • [C. Specimens p3](#) • [D. Function Tables p4](#) • [D. Adverbs & Conjunctions p8](#) • [E. Explicit Definitions p9](#) • [H. Gerunds p12](#) • [B. Merge & Amend p14](#) • [B. Partition p19](#) • [C. Special Matrices & Lists p20](#) • [A. Permutations p24](#) • [C. Parity & Symmetry p26](#) • [B. Linear Vector Functions p33](#) • [C. Polynomials & Rational Functions p34](#) • [A. Finance p47](#)
- x: [A. Numbers & Counting p27](#) • [A. Inverse p44](#) • [A. Finance p47](#) • [B. Date & Time p52](#)

Y

- y. [C. Specimens p3](#) • [D. Adverbs & Conjunctions p8](#) • [E. Explicit Definitions p9](#) • [A. Intervals p15](#) • [B. Locating & Selecting p16](#) • [C. Special Matrices & Lists p20](#) • [A. Numbers & Counting p27](#) • [B. Grids p28](#) • [C. Polynomials & Rational Functions p34](#) • [B. Duality p45](#) • [A. Finance p47](#) • [A. Inside Boxes p48](#)
- year(s) [A. Conventions p1](#) • [B. Duality p45](#) • [A. Finance p47](#) • [B. Date & Time p52](#)
- yield [C. Parity & Symmetry p26](#) • [C. Polynomials & Rational Functions p34](#) • [F. Geometry p37](#)

Z

- zeros [C. Specimens p3](#) • [B. Locating & Selecting p16](#)