ZigZag
A CHAUTAUQUA ON J
DR. JOSHUA ECKROTH
STETSON UNIVERSITY
Syntax
- Statements are exactly one line long
- Numbers may be written 2, _2, 0.5, 5, 1e3, 2j5, 2r7, 16b1f, 2p1, _, __
- Adjacent numbers make a list: 3 5 2e3 7
- Strings have single-quotes: 'hello'
- Everything else is a built-in or user-defined word:
  - Verbs: +, -, %, etc.
  - Adverbs: ~, /, }, etc.
  - Conjunctions: ^:, ", @, &, etc.
  - Others: =., =:, NB., etc.
You do not need to trouble yourself with the distinction between integers, floats, and complex numbers. If it’s a number, J will handle it properly.

J for C Programmers
Evaluation Rules
Forget the table of operator precedence!

J for C Programmers
RTL Evaluation:

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

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

Plus

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

Verb
Negate
-\( y \) is the negative of \( y \). That is, it is defined as \( 0 - y \).
Thus, \(-2 \) is \(-2\).

Minus
\(-\) is defined as in elementary arithmetic, and is extended to complex numbers as usual.

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

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

Verb
<table>
<thead>
<tr>
<th>Reciprocal</th>
<th>Divided by</th>
</tr>
</thead>
<tbody>
<tr>
<td>% y is the reciprocal of y, that is, ( \frac{1}{y} ).</td>
<td>x % y is division of x by y as defined in elementary math, except that 0%0 is 0.</td>
</tr>
</tbody>
</table>
Monads & Dyads
y is right arg

Reciprocal
% y is the reciprocal of y, that is, 1/y.

y is right arg

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

x is left arg,
y is right arg

Verb
Monad vs. Dyad %:

2%7
0.285714
7%2
3.5
%7
0.142857
%2
0.5
Challenge!

Compute the arithmetic mean of 4 and 7.

Find three solutions using all of these cards simultaneously:
Self-Classify

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

Equal

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

Verb
Magnitude       Residue
\lfloor y \rfloor = \%:y+y . The familiar use of residue is in determining the remainder on dividing a non-negative integer by a positive.
Open is the inverse of box, that is, \( \triangleright \triangleright \triangleright \triangleright \) is \( \triangleright \triangleright \triangleright \triangleright \) is tolerably larger than \( \triangleright \triangleright \triangleright \triangleright \). When applied to an open array (that has no boxed elements), open has no effect. Opened atoms are brought to a common shape.

Verb
~: _ 0 0

Nub Sieve

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

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

Verb
Not

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

Less

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

Compute whether a number, say 8, is even (gives 1 if even, 0 otherwise).
Find four solutions using all of these cards simultaneously:
Arrays
Any variable can be an array. As for what the type and dimensioning is: you assigned the variable, didn't you? It contains whatever you put into it, a number, a string, an array, a structure... J will remember. If your program logic requires you to find out the current attributes of a variable, J can tell you.

*J for C Programmers*
Tally

\#y is the number of items in y.

Copy

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

```plaintext
a =: 5 3 7 9
s =: 'hello'
```
In J, every operator has a loop built in.

J for C Programmers
Automatic element-by-element operations:

\[
\begin{array}{cccccc}
1 & 2 & 3 & + & 4 & 5 & 6 \\
5 & 7 & 9 \\
1 & + & 4 & 5 & 6 \\
5 & 6 & 7 \\
1 & 2 & + & 4 & 5 & 6 \\
| \text{length error} \\
| 1 & 2 & + & 4 & 5 & 6 \\
\end{array}
\]
The shape of i.y is ly, and its atoms are the first */ly non-negative integers. A negative element in y causes reversal of the atoms along the corresponding axis.

Verb

Integers

Index Of

If rix is the rank of an item of x, then the shape of the result of x i. y is (-rix)\{y . Each atom of the result is either #x or the index of the first occurrence among the items of x of the corresponding rix-cell of y.
# _ 1 _

**Tally**


#y is the number of items in y.

**Copy**

If the arguments have an equal number of items, then x###y copies +/x items from y, with i###x repetitions of item i###y.

Otherwise, if one is an atom it is repeated to make the item count of the arguments equal.

**Verb**
<table>
<thead>
<tr>
<th>i.4</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3+i.4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3#i.4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>0 0 0 1 1 1 2 2 2 3 3 3</td>
<td>#3#i.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3##i.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 4 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 4 4 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
? 0 0 0

Roll

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

Deal

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

Verb
Challenge!

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

Find four solutions using all of these cards simultaneously:
Adverbs
A noun holds data; a verb operates on one or two nouns to produce a result which is a noun; an adverb operates on one noun or verb to produce a derived entity.

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

Adverb
1 + 2 + 3 + 4 + 5
15
+/1 2 3 4 5
15
*/1 2 3 4 5
120
New version of “arithmetic mean”:

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

Reflexive

\[ u \sim y = y \uparrow y \uparrow . \]

For example, \( ^\sim \)

3 is 27, and +/-

\( ^\sim \) i. n is an

addition table.

Passive

\[ ^\sim \text{ commutes or} \]

crosses

connections to

arguments: \( x \uparrow u \sim y = y \uparrow u \uparrow x \).
Catalogue

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

From

If \( x \) is an integer in the range from \(-n\) to \( n-1 \), then \( x\{y \text{n} \text{l} x \text{ from } y \text{.} \)

Verb
Challenge!

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

Find two solutions using all of these cards simultaneously:
Hooks & Forks
Hooks

Monadic

\[ fg(y) = y \, f \, (g \, y) \]
\[ -\% \, y = y - (\% \, y) \]

Dyadic

\[ x \, fg \, y = x \, f \, (g \, y) \]
\[ x \, -\% \, y = x - (\% \, y) \]
f =: monad : '0=2|y'  NB. checks that y is even
i.10
0 1 2 3 4 5 6 7 8 9
f i.10
1 0 1 0 1 0 1 0 1 0

NB. use # operator to keep only evens
(f i.10) # i.10
0 2 4 6 8

NB. use a hook
(#~f) i.10  NB. i.e., i.10 #~ (f i.10)
0 2 4 6 8  NB. i.e., (f i.10) # i.10
Forks  \( fgh \)

**Monadic**
\[
\begin{align*}
  fgh \ y &= (f \ y) \ g \ (h \ y) \\
  -+\% \ y &= (- \ y) + (\% \ y)
\end{align*}
\]

**Dyadic**
\[
\begin{align*}
  x \ fgh \ y &= (x \ f \ y) \ g \ (x \ h \ y) \\
  x \ -+\% \ y &= (x - y) + (x \ % \ y)
\end{align*}
\]
Challenge!

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

Find one solution for each kind of mean using all of these cards simultaneously:
Conjunctions
J uses the names x, y, u, v, m, and n to represent arguments to verbs and other entities. You should avoid using these names for other purposes.

*J for C Programmers*
Bond

\( m \& v \ u \& n \ \_ \ 0 \_ \)

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

Conjunction
f =: 5&+
f 1 2 3 4 5
6 7 8 9 10

g =: -&5
g 1 2 3 4 5
_4 _3 _2 _1 0
**Compositions**

\[ u \& v \ yeq u \ yeq v \]

Thus, for example, \( 3 + 4 = 7 \) and \( 3 + 0 = 3 \). For the negation, \( \neg 4 = 1 \). Moreover, the monads \( u \& v \) and \( u \& v \) are equivalent.

**Conjunction**

\[ u \& v \ yeq v \ yeq u \]

Thus, for example, \( 3 \& 4 = 12 \) and \( 0 \& 4 = 0 \). For the negation, \( \neg 4 = 0 \). Moreover, the monads \( u \& v \) and \( u \& v \) are equivalent.
f =: -&% 
f 1 2 3 4 5  
_1 _0.5 _0.333333 _0.25 _0.2 
10 f 1 2 3 4 5  
_0.9 _0.4 _0.233333 _0.15 _0.1 

NB. i.e., -(%y) 

10 g 1 2 3 4 5  
0 1.5 2.66667 3.75 4.8 
10 g 1 2 3 4 5  
9 9.5 9.66667 9.75 9.8 

NB. i.e., (%x)-(y) 

NB. compare with hook 
NB. i.e., y-(y) 
NB. i.e., x-(y)
At

\( u@v y = u \circ v y \). For example, \( y \). For example, \( u@v \circ v \).

\( +@- 7 = _{14} \)  \( +@- 7 = _{8} \)

(double the (double the
negation). difference).

Moreover, the monadic uses of \( u@v \) and \( u\circ v \) are equivalent.

Conjunction
\[
f =: -@:\% \quad \text{NB. same as \& in monadic form}
\]
\[
\begin{array}{c}
f 1 2 3 4 5 \\
\_1 \_0.5 \_0.333333 \_0.25 \_0.2
\end{array}
\]
\[
\begin{array}{c}
10 f 1 2 3 4 5 \\
\_10 \_5 \_3.333333 \_2.5 \_2
\end{array}
\quad \text{NB. i.e., } -(x \% y)
\]
\[
\begin{array}{c}
f =: +/@:\% \\
f 1 2 3 4 5 \\
2.28333
\end{array}
\quad \text{NB. i.e., } +/(% \ y) \quad \text{NB. i.e., sum reciprocals of } \ y
\]
**Challenge!**

Define the $L^1$, $L^2$, and $L^\infty$ norms.

$L^1 = \text{sum}(\text{absval}(x_i))$ for all $x_i$

$L^2 = \sqrt{\text{sum}(x_i^2)}$ for all $x_i$

$L^\infty = \text{max}(x_i)$ for all $x_i$

Find one solution for each norm using all of these cards simultaneously:
Atoms, Lists, Tables, et al.
Ravel                  Append

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

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

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

Verb

Laminate

An atomic argument in x,;y is first reshaped to the shape of the other (or to a list if the other argument is also atomic); the results are then itemized and catednated, as in (,;x), (,;y).
pt1   ::=  5.1  3.5  1.4  0.2
pt2   ::=  4.9  3.0  1.4  0.2
pt3   ::=  4.7  3.2  1.3  0.2
pt1 , pt2 , pt3
5.1  3.5  1.4  0.2  4.9  3  1.4  0.2  4.7  3.2  1.3  0.2
pt1 , pt2 ,: pt3
5.1  3.5  1.4  0.2
4.9   3  1.4  0.2
4.7  3.2  1.3  0.2
Shape Of

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

Shape

The shape of $ x $ $ y $ is $ x $, $ s i y $ where $ s i y $ is the shape of an item of $ y $;

$x $ $ y $ gives a length error if $ y $ is empty and $ x $, $ s i y $ does not contain a zero.

Verb
pt1, pt2, pt3
5.1 3.5 1.4 0.2 4.9 3 1.4 0.2 4.7 3.2 1.3 0.2
$ pt1, pt2, pt3
12
   pt1, pt2,: pt3
5.1 3.5 1.4 0.2
4.9 3 1.4 0.2
4.7 3.2 1.3 0.2
   $ pt1, pt2,: pt3
3 4
Rank
The rank of a noun is the count of its axes. An atom has rank 0, a list rank 1, a table rank 2, and an array with 5 axes has rank 5.

_J Primer_
$ pt1, pt2, pt3
12
#$ pt1, pt2, pt3 NB. #$ computes rank 1
$ pt1, pt2,: pt3
3 4
#$ pt1, pt2,,: pt3
2
NB. etc.
Ask the rank of a verb.
If you don’t know the rank of a verb, you don’t know the verb. Using a verb of unknown rank is like wiring in a power-supply of unknown voltage—it will do something when you plug it in; it might even work; but if the voltage is wrong it will destroy what it’s connected to. Avoid embarrassment! Know the rank of the verbs you use.

_J Primer_
"u"n

Rank

The verb "u"n applies u to each cell as specified by the rank n. The full form of the rank used is 3<&1.n. For example, if n=2, the three ranks are 2 2 2, and if n=2 3, they are 3 2 3. A negative rank is complementary: "u"(-r) y is equivalent to "u"(0>.(#y)-r)"_ y.

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

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

100 200 300 +"_ 0 (1 2 3 4)
101 201 301
102 202 302
103 203 303
104 204 304
Show how a verb would be applied.

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

8 divide 10

4 8 divide 10 20

(4 8) divide 10 20
```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4 8</td>
<td>(divide&quot;0) 10 20</td>
</tr>
<tr>
<td>4 divide 10</td>
<td>8 divide 20</td>
</tr>
<tr>
<td>4 8 (divide&quot;1) 10 20</td>
<td></td>
</tr>
<tr>
<td>(4 8) divide 10 20</td>
<td></td>
</tr>
</tbody>
</table>
### 4 (divide"0) i.2 3

<table>
<thead>
<tr>
<th>4 divide 0</th>
<th>4 divide 1</th>
<th>4 divide 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 divide 3</td>
<td>4 divide 4</td>
<td>4 divide 5</td>
</tr>
</tbody>
</table>
4 \(\text{divide"1\)}\ i.2\ 3

| 4 \text{divide } 0 1 2 | 4 \text{divide } 3 4 5 |

4 \(\text{divide"\_\)}\ i.2\ 3

| 4 \text{divide } 2 3$(0 1 2) (3 4 5)$ |
\[(i.2 \ 3) \ (\text{divide"}0 \ 0) \ i.2 \ 3\]

<table>
<thead>
<tr>
<th></th>
<th>0 divide 0</th>
<th>1 divide 1</th>
<th>2 divide 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3 divide 3</td>
<td>4 divide 4</td>
<td>5 divide 5</td>
</tr>
</tbody>
</table>
\[(i.2 \ 3) \ (\text{divide}^"1 \ 0) \ i.2 \ 3\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>((0 \ 1 \ 2)) divide 0</td>
<td>((0 \ 1 \ 2)) divide 1</td>
<td>((0 \ 1 \ 2)) divide 2</td>
</tr>
<tr>
<td>((3 \ 4 \ 5)) divide 3</td>
<td>((3 \ 4 \ 5)) divide 4</td>
<td>((3 \ 4 \ 5)) divide 5</td>
</tr>
</tbody>
</table>
(i.2 3) (divide"1 1) i.2 3

| (0 1 2) divide 0 1 2 | (3 4 5) divide 3 4 5 |
\[
\begin{array}{c}
(i.2 \ 3) (\text{divide"1 \ _\ ) i.2 \ 3}
\end{array}
\]

\[
\begin{array}{c|c|c}
(0 \ 1 \ 2) \ & \text{divide} & 2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) \\
(3 \ 4 \ 5) \ & \text{divide} & 2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5)
\end{array}
\]

\[
\begin{array}{c}
(i.2 \ 3) (\text{divide"\_ \ 0\ ) i.2 \ 3}
\end{array}
\]

\[
\begin{array}{c|c|c|c}
(2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) \ ) \ & \text{divide} & 0 & (2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) \ ) \ & \text{divide} & 1 & (2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) \ ) \ & \text{divide} & 2 \\
(2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) \ ) \ & \text{divide} & 3 & (2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) \ ) \ & \text{divide} & 4 & (2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) \ ) \ & \text{divide} & 5
\end{array}
\]

\[
\begin{array}{c}
(i.2 \ 3) (\text{divide"\_ \ 0\ ) i.2 \ 3}
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
0 \ & \text{divide} & 2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) & 1 \ & \text{divide} & 2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) & 2 \ & \text{divide} & 2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5)
3 \ & \text{divide} & 2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) & 4 \ & \text{divide} & 2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) & 5 \ & \text{divide} & 2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5)
\end{array}
\]

\[
\begin{array}{c}
(i.2 \ 3) (\text{divide"\_ \ _) i.2 \ 3}
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
(2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5) \ ) \ & \text{divide} & 2 \ 3\$(0 \ 1 \ 2) \ (3 \ 4 \ 5)
\end{array}
\]
In J, every operator has a loop built in.

J for C Programmers
Coding Challenge!

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

Compute the column sums of a table.

<table>
<thead>
<tr>
<th></th>
<th>1.4</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>40</td>
<td>44 48 52 56</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>1.2</th>
<th>3</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>8.5</td>
<td>14.5</td>
<td></td>
</tr>
<tr>
<td>20.5</td>
<td>26.5</td>
<td>32.5</td>
<td></td>
</tr>
</tbody>
</table>
Recursion
The verb \( u \) is applied \( n \) times. An infinite power \( n \) produces the limit of the application of \( u \). If \( n \) is negative, the obverse \( u^{\wedge:-1} \) is applied \( n \) times.

**Power**

\[
x \ u^{\wedge:n} \ y \ = \\
x \& u^{\wedge:n} \ y
\]

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

\[
u \ ^{\wedge}: \ v \ y \ = \\
u^{\wedge:(v \ y)} \ y \\
x \ u^{\wedge}: \ v \ y \ = \\
x \ u^{\wedge:(x \ v \ y)} \ y
\]
vals=:4 3 2 $ 24?100
>./vals
96 95
70 85
60 89
>././vals
96 95
>./././vals
96
>./^:3 vals NB. recursive application, 3 times
96
>./^:_ vals NB. 'limit' of >./, works on any shape
96
Recursive “Larger of” >.
NB. interesting cases of u^:n

NB. do-once
(>:^:1) 10  NB. increment once
11

NB. do-many
(>:^:5) 10  NB. increment five times
15

NB. don’t
(>:^:0) 10  NB. returns y
10
NB. interesting cases of u^n

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

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

NB. do-infinitely-many-times
(cos^:_1) 1  NB. compute y=cos(y)
0.739085
\[ u^n \quad u^v \]

**Power**

The verb \( u \) is applied \( n \) times. An infinite power \( n \) produces the limit of the application of \( u \). If \( n \) is negative, the obverse \( u^\neg 1 \) is applied \( n \) times.

\[
x^u^v = \frac{x}{u^v}
\]

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

\[
\begin{align*}
u^v &= v y \\
u^\neg(v y) &= (v y)
\end{align*}
\]

**Conjunction**

\[
x u^v = x v y y
\]
NB. highly interesting cases of u^:v

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

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

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

1 2 3 4 5 myfunc if 1 0 0 1 0
2 2 3 5 5 5
NB. more fancy

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

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

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

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

dowhile =: conjunction define
(u^:(v@:])^:"0) y
:
x (u^:(v@:])^:"0) y
)

NB. continue doubling a number until it’s >= 100
2 * dowhile (100&>) 1 3 5 7 9 11
128 192 160 112 144 176
NB. defining “if” and “dowhile” as a J adverb (if) NB. and conjunction (dowhile) gives us the benefits NB. of automatic looping:

(2&* if even) i. 2 3 2
0 1
4 3
8 5
12 7
16 9
20 11
NB. defining “if” and “dowhile” as a J adverb (if)
NB. and conjunction (dowhile) gives us the benefits
NB. of automatic looping:

(2&* dowhile (100&>)) i. 2 3 2

0 128
128 192
128 160

192 112
128 144
160 176
Coding Challenge!

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

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

1  ______  90  70  80  75  
   NB. drop 1 score
   81.6667  
2  ______  90  70  80  75  
   NB. drop 2 scores
   85  
3  ______  95  60  0  55  80  0  90  
   NB. drop 3 scores
   78.333
Big Data
You may at first worry that you're using too much memory, or that you might misuse the processor's caches; get over it. Apply verbs to large operands.

*J for C Programmers*
Jd, the J relational database.

load 'jd'

NB. create a database folder
jdadmnx'bbjd'

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

NB. create a "cdefs" file that describes the dataset
jd'csvcdefs /h 1 backblaze-2013-2016.csv'
NB. load the backblaze data (60mil rows)
jd'csvrd backblaze-2013-2016.csv backblaze'

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

NB. show the db schema
jd'info schema'

<table>
<thead>
<tr>
<th>table</th>
<th>column</th>
<th>type</th>
<th>shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>backblaze</td>
<td>date</td>
<td>edate</td>
<td>_</td>
</tr>
<tr>
<td>backblaze</td>
<td>serial_number</td>
<td>byte</td>
<td>15</td>
</tr>
<tr>
<td>backblaze</td>
<td>model</td>
<td>byte</td>
<td>23</td>
</tr>
<tr>
<td>backblaze</td>
<td>capacity_bytes</td>
<td>int</td>
<td>_</td>
</tr>
<tr>
<td>backblaze</td>
<td>failure</td>
<td>boolean</td>
<td>_</td>
</tr>
<tr>
<td>backblaze</td>
<td>smart_187_raw</td>
<td>int</td>
<td>_</td>
</tr>
</tbody>
</table>
NB. read some data
jd'reads date,capacity_bytes from backblaze'

...  
| 2013-04-10 | 3000592982016 |
| 2013-04-10 | 3000592982016 |
| 2013-04-10 | 3000592982016 |
| 2013-04-10 | 3000592982016 |
| 2013-04-10 | 2000398934016 |
| 2013-04-10 | 2000398934016 |
| 2016-12-29 | 322056190273242112 |
| 2016-12-30 | 322056190273242112 |
| 2016-12-31 | 322056190273242112 |

NB. save result
capacity_by_day =: (...above...
NB. results are "boxed" (each box can hold a different type)
NB. so let's unbox just the capacities

```
capacities =. ,/ > 1 1 { capacity_by_day
```

```
capacities
54041214222213120 54041214222213120 54041214222213120
53861178805862400 55657532182339584 55657532182339584
55792558378819584 55796559165849600 55796559165849600
55796559165849600 55796559165849600 55795058863939584
55795058863939584 55930085548130304 ...
```

```
$ capacities
1360
```

```
NB. find min, max capacity

<./ capacities
_8946789308003949568

>./ capacities
1282429594186119424
```
NB. avg capacity
(+/-%) capacities
3.55142e12

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

<table>
<thead>
<tr>
<th>aggs</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg</td>
</tr>
<tr>
<td>count</td>
</tr>
<tr>
<td>countunique</td>
</tr>
<tr>
<td>first</td>
</tr>
<tr>
<td>last</td>
</tr>
<tr>
<td>max</td>
</tr>
<tr>
<td>min</td>
</tr>
<tr>
<td>sum</td>
</tr>
</tbody>
</table>

jd'reads avg capacity_bytes from backblaze

<table>
<thead>
<tr>
<th>capacity_bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.55142e12</td>
</tr>
</tbody>
</table>

NB. 5sec
NB. compute overall failure rate
failures =: ,/>0{1{ jd'reads failure from backblaze'
$ failures
59654783
+/failures  NB. total failures
5806
overallAFR =: 365*100*(+/failures)%(#failures)
overallAFR
3.55242
NB. compute failures by year
dates =: >0{1{ jd 'reads date from backblaze'
$ dates
59654783 10      NB. ugh, dates are strings now
   2 {. dates
2013-04-10
2013-04-10
2013-04-10
   (4&{."1) 2 {. dates  NB. keep only first four digits
2013
2013

NB. 0". converts strings to numbers
years =: 0 ". (4&{."1) dates
$ years
59654783
NB. ok, we have years
NB. how do we sum failures per year?

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

```
m/.
```

**Oblique**

u/.y applies u to each of the oblique lines of a table y. m/.y applies successive verbs from the gerund m to the oblique lines of _2_-cells of y, extending m cyclically as required.

**Key**

x w/.y -- (<<) u@# y, that is, items of x specify keys for corresponding items of y and u is applied to each collection of y having identical keys. x m/.y applies successive verbs from the gerund m to the collections of y, extending m cyclically as required.
NB. ok, we have years
NB. how do we sum failures per year?

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

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

NB. hence:
driveFailuresYears =: years +//. failures
NB. 1sec
driveFailuresYears
740 2206 1429 1431
NB. likewise:
driveDays =: years #/. failures
driveDays
5091501 12582414 17509251 24471617

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

NB. hence hence:
yearlyAFR =: 365*100*driveFailuresYears%driveDays
uniqueYears ,: yearlyAFR
2013 2014 2015 2016
5.30492 6.39933 2.97891 2.13437
### The big slide with timing data.

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

Note, original CSV data was 3.2 GB.
Visualization
load 'addons/graphics/fvj4/dwin.ijs'
_1_1 1 1 dwin 'fun'   NB. open window, -1:1 axes
1 o. 1r6p1            NB. sin(1/6 pi^1)
0.5
2 o. 1r6p1            NB. cos(1/6 pi^1)
0.866025

NB. define x,y coordinates of a circle with 200 pts
circ =: |: 2 1 o./ 1r100p1*i.200
1   0
0.999507  0.0314108
0.998027  0.0627905
0.995562  0.0941083
...
NB. plot a circle with a color
255 0 255 dpoly circ

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

NB. plot four circles with individual colors
colr4 dpoly circ4
NB. generate some exponential values
\((^0.1 * i.80) \% 100\)
0.01 0.0110517 0.012214 0.0134986 ...

NB. generate some x,y values for a symmetric curve
\(xs =: ((i.180) * \%180) - 0.5\)
\(ys =: ((|.) (^0.075*i.90)), (^0.075*i.90))\%400\)
\(curve =: |: xs ,: ys\)
\(curve\)
\(-0.5\quad 1.98087\)
\(-0.494444\quad 1.83774\)
\(-0.488889\quad 1.70495\)
...
```plaintext
curve
_0.5 1.98087
_0.494444 1.83774
_0.488889 1.70495
...
_0.0111111 0.00269471
_0.00555556 0.0025
0 0.0025
0.00555556 0.00269471
0.0111111 0.00290459
...

0.483333 1.70495
0.488889 1.83774
0.494444 1.98087

NB. plot the curve
255 0 255 dline curve
```
Rotation can be defined as matrix multiplication of a vector representing the point: \([x, y, 1]\) and a rotation matrix:

\[
M = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

After multiplying \(M^* [x, y, 1]^\top\), the result will be \([x', y', z]\); divide \(z\) out of \(x'\) and \(y'\) to get final coordinates. In our case, however, \(z\) will always be 1.

NB. define matrix mult.
\(mp =: +/ . *\)

NB. allow simple trig names
load 'trig'

NB. define rotation matrix
\(rotm =: (\cos, \sin, 0:),\)
\((-@\sin, \cos, 0:),:\)
\((0:, 0:, 1:))\)

\(rotm 1r4p1\)
\[
\begin{bmatrix}
0.707107 & 0.707107 & 0 \\
-0.707107 & 0.707107 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]
Likewise, a translation matrix is defined as:

\[
M = \begin{bmatrix}
1 & 0 & x_{\text{trans}} \\
0 & 1 & y_{\text{trans}} \\
0 & 0 & 1
\end{bmatrix}
\]

Translation & rotation matrices can be combined with matrix multiplication, e.g.,

\[
\text{TransM} \ast \text{RotM} \ast [x,y,1]^T
\]
NB. add extra z=1 dimension to curve
curve =: curve,.1

NB. draw a translated, rotated curve
dclear''
255 0 255 dline curve mp (trans 0.5 0.2) mp (rotm 1r2p1)
NB. define a verb for drawing curve with translate + rotate
NB. use dyad form, x=translate, y=rotate
dcurveRot =: dyad define
255 0 255 dline curve mp (trans x) mp (rotm y)
)
dclear''
0.5 0.2 dcurveRot 1r2p1
0.5 0.2 dcurveRot 2r2p1
0.5 0.2 dcurveRot 3r2p1
0.5 0.2 dcurveRot 4r2p1
NB. draw many!
dclear''
(0 0.5 & dcurveRot)"0 (1r10p1 * i.20)
NB. unrotate by pi as the last step before drawing,
NB. and eliminate x-coordinate of translation parameter
dcurveRot =: dyad define
255 0 255 dline curve mp (rotm _1p1) mp (trans 0,x) mp (rotm y) )
dclear''
1 dcurveRot"0 (1r10p1 * i.20)
NB. generate 4 translations 0.25 apart
translations =: 0.25 * i.4
NB. generate 14 rotations 1/7 pi apart
rotations =: 1r7p1 * i.14
NB. draw each rotation/translation pairing (48 pairings)
dclear''
(48$translations) dcurveRot"0 (48$rotations)
A scaling (zooming) matrix can be defined as:

\[
M = \begin{bmatrix}
s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1
\end{bmatrix}
\]

NB. scaling matrix as verb

```plaintext
scale =: monad define
(y, 0 0), (0, y, 0),: 0 0 1
)
```

```
scale 5
5 0 0
0 5 0
0 0 1
```

```
(scale 5) mp (trans 5 2) mp (rotm 1r4p1)
3.53553 3.53553 0
_3.53553 3.53553 0
2.12132 4.94975 1
(3 2 1) mp (... above ...) 
5.65685 22.6274 1
```
NB. update dcurveRot to use scale based on translation size

dcurveRot =: dyad define
m =. (scale x) mp (rotm _1p1) mp (trans x) mp (rotm y)
255 0 255 dline curve mp m
)
dclear''
(48$translations) dcurveRot"0 (48$rotations)
NB. update dcurveRot to use (scale 0.185\times x) and
NB. obsessively fine-tune numbers to arrive at:
translations =: 0.09*i.50
rotations =: 1r22p1+1r11p1*i.22
(1100\$translations) dcurveRot"0 (1100\$rotations)
NB. next we need to set the color based on x;
NB. we’ll use HSV scale for gradual color changes

dcurveRot =: dyad define
  m =. (scale 0.185*x) mp (rotm _1p1) mp (trans 0,x) mp (rotm y)
  (hsv2rgb ((360|90000*x^0.015),1.0 1.0 1.0)) dline curve mp m
)
NB. MOAR CURVES
translations =: 0.005 * i.400
rotations =: 1r11p1*i.22
(22#translations) dcurveRot"0 (8800$rotations)
NB. add those extra curves rotated \( \pi/22 \) and call it a day