Journal of J

An open Access journal

An interdisciplinary journal on J programming language and aplications in science and technology

R.E. Boss _____ *n-step Fibonacci sequences in J*

N. MacKenzie _____ *Notes on Equivalence. Between some Boolean Arrays of Different Rank.*

L. Alvord _____ *Using Color to See Patterns in Numeric Tables.*

ISSN: 2174-9280

Vol. 4, N0. 1 March 2015 www.journalofj.com

n-step Fibonacci sequences in J

R.E. Boss

1. Introduction

An *n*-step Fibonacci number is a sequence of numbers where every item is the sum of the *n* preceding numbers. According to Mathworld the initial terms are prescribed too, but actually they are a free choice. The usual Fibonacci numbers 1,1,2,3,5,8,... are thus the 2-step Fibonacci numbers.

Notice that the Lucas numbers 2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, ... are also a 2-step Fibonacci sequence, but with different starting numbers, 2,1 instead of 1,1.

Not only the Fibonacci numbers but also the Lucas numbers are well known in the OEIS.

The normal way of generating *n*-step **fibseqs**, as we will abbreviate Fibonacci sequences, in J is of course the next expression

```
F0=: ], [:+/ ({.~-)~
```

such that the first part of the 2-step sequence can be calculated by

```
   2 F0^:(3) 1 2
1 2 3 5 8
```

In the essay on Fibonacci sequence Hui gathered some 10 ways of generating *a special member* of this most well-known 2-step sequence, and not so much *all numbers* up to a certain amount.

Some of the solutions presented by Hui can be used for generating the sequence as well. Among those I prefer Matrix Power

```
F1=: [:, (1 1,:1 2)(+/ . *)^:(<@]`(1x 1"_)) ]
```

It generates a sequence of length `y`.

Another one I made up my self, but performs less than I would have thought, is

```
F2=: 3 :'(,[:(+/ . * {:) 2[\ ])^:y 1 1x'
```

which generates a sequence of length `2^y` from the normal Fibonacci sequence.

Below we will give a fast way of generating (the first part of) an *n*-step fibseq.

2. (most) Efficient way generating (2-step) Fibonacci sequence

Last summer I found a really fast and, what's more, elegant solution to generate a lot of terms of an *n*-step fibseq with extended precision.
Look at the first 6 numbers in the Fibonacci sequence 1,1,2,3,5,8 and make the _2 infix.

```
   _2[\1 1 2 3 5 8
1 1
2 3
5 8
```

Now I would like to generate the next row of this matrix. The first number of that row is easy, since it is 13 which is `+/5 8` . But for the second I have to add 8 again.

But wait, if I do `+/\.5 8` instead, I get

```
   +/\. 5 8
13 8
```

And all of a sudden it appears to be rather straightforward to generate 13 21 as next pair.

```
   +/\+/\. 5 8
13 21
```

The rest, of course, is easy. Let

`F3=:[:, ([:+/\ +/\.)^: (<@]`(1x 1"_))` then

```
   F3 10
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

If we compare the performance of these solutions we have to be aware of the fact that not all sequences are exactly equal in size since

```
   #L:0(2 F0^:(2^12) 1 1x);( F1 2^11);(F2 12);F3 2^11
+----+----+----+----+
|4098|4096|4097|4096|
+----+----+----+----+
```

However, with some amendments we get

```
   2 -:/\(2 F0^:(_2+ 2^12) 1 1x); (F1 2^11); (}:F2 12); F3 2^11
1 1 1
```

The performance results are

```
   >10 ts L:0'2 F0^:(_2+2^12) 1 1x';'F1 2^11';'}:F2 12';'F3 2^11'
  0.63396064 5622528
 0.018546692 9186304
  0.19924922 5956352
0.0055577757 5964544
```

place	rlprf	rlt	rls	verbs
0	1.00	1.00	1.06	F3
1	5.45	3.54	1.63	F1
2	36.83	36.88	1.06	F2
3	111.78	118.58	1.00	F0

so we come to the following ranking, with relative performance, relative timing and relative space, respectively, and performance being the product of timing and space [1].

Conclusion is that F3 is the fastest and overall the most efficient and F0 is the leanest. Notice that for non-extended sequences the performance figures can be much different.

3. *n*-step fibseqs

The question is to generalize these verbs to the case of the *n*-step fibseqs with `2<n`.

3.1.FS0

This is the easiest to generalize since it is just altering the left hand variable.

```
   5 F0^:(10) 1 2 3 4 5
1 2 3 4 5 15 29 56 109 214 423 831 1633 3210 6311
```

to add 30 items to your initial sequence.

But we want the left hand variable to be the number of iterations, and so the step size is the length of the start sequence. So we get

`FS0=:F0^:(#@]`[`])` or better `FS0=: [], [: +/ ({.~ -)~)^: (#@]`[`])`

3.2.FS1

To generalize `F1`, we have to construct first the analogue of the matrix $\begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$. Fortunately, my alter ego has published an article recently [2] where the general matrix for arbitrary *n*-step fibseqs was derived. It can be produced by the following J-expression for the 5-step fibseq matrix.

[1] see Addendum for the verbs for this table
[2] to appear

```
   +/\"1@(|.\)@(1,2&^@i.@<:)5
1  1  1  1  1
1  2  2  2  2
2  3  4  4  4
4  6  7  8  8
8 12 14 15 16
```

Although this is the most efficient way, this matrix generates floating numbers, as ^ does. We prefer the matrix to be of integer type to generate sequences of the same data type, so we will use another expression, which is slightly less efficient, but only for high values.

```
   datatype 3 (fsm=: +/\"1@(|.\)@(*/\)@(1 1, 2#~ <:@<:)) 5
integer
```

Now the matrix is known, F1 can be generalized to FS1 as follows, where x is the number of iterations and y the start sequence.

```
  FS1=: [:,(+/ .*)^:(fsm@#@]`(<@[)`])
   3 FS1 1 2 3 4 5
1 2 3 4 5 15 29 56 109 214 423 831 1633 3210 6311
```

Some remarks have to be made.

- The length of the sequence to be generated is, in this case, 5*10=50, so *n* times the number of iterations, where *n* is the step size.
- We could determine a specific item of the 5-step fibseq by the slightly altered expression
  ```
    ,.20 (+/ .*)^:(fsm@#@]`[`]) 1 2 3 4 5x
  111510825581360911080044717605
  219224510918558760467879809583
  430984040669809542154739855709
  847292314777158206670312154168
 1665732832159425674998800136909
  ```
  which give items 100 – 104 of the 5-step fibseq.
- If we want extended precision, we could as we saw in the former remark, start with a sequence of extended precision
- As the original (2-step) Fibonacci sequence appears to be unique modulo shifts, there are however a lot of different possibilities for *n*-step Fibonacci sequences, see oeis.org. and the paper referred to above.

[3] see Addendum for definition of datatype

3.3. FS2

The second verb to be generalized is F2 and this is a lot less trivial than you should suspect. Because simply changing the 2's in F2 in 5 e.g. doesn't give you what you want since

```
   (,[:(+/ . * {:) 5[\ ])^:3[1 0 0 0 0
1 0 0 0 0 1 0 1 0 1 0 2
```

is obviously wrong from the 7th item on, which should be 1.

According to the article referenced earlier, each *n*-step fibseq is the linear combination of *n* primitive fibseqs, generated by $\left(\underbrace{1,0,\cdots,0}_{n}\right);\left(\underbrace{0,1,0,\cdots,0}_{n}\right);\cdots;\left(\underbrace{0,\cdots,0,1}_{n}\right)$. So if we have the start sequence *a*,*b*,*c* for a 3-step fibseq, then the next numbers are *a*+*b*+*c*, *a*+2*b*+2*c*, 2*a*+3*b*+4*c*, 4*a*+6*b*+7*c*, 7*a*+11*b*+13*c*, ... and the coefficients of *a*, *b* and *c* are just (in this case) the three 3-step fibseqs

```
   |:(, [:+/ _3&{.)^:15 =/~ i.3
1 0 0 1 1 2 4  7 13 24 44  81 149 274 504  927 1705 3136
0 1 0 1 2 3 6 11 20 37 68 125 230 423 778 1431 2632 4841
0 0 1 1 2 4 7 13 24 44 81 149 274 504 927 1705 3136 5768
```

And the same holds for n-step sequences in general.

So, in general, we have to generate first the *n*-step sequences (from the rows of the `n*n` unit matrix) to generate any *n*-step sequence at all. And this should be done in the F2 way.

Based on the example above and the look of F2, we come to the 3-step generator

```
   |:([, {: (+/ .*)"(1 _1) 3[\])^:4 (}.,+/) =/~ i.3
0 0 1 1 2 4  7 13 24 44  81 149 274 504  927 1705 3136  5768
1 0 1 2 3 6 11 20 37 68 125 230 423 778 1431 2632 4841  8904
0 1 1 2 4 7 13 24 44 81 149 274 504 927 1705 3136 5768 10609
```

Here first the unit matrix is made, but with the sum of the rows added and the first row deleted to get the first 3 columns. Then each time we take the last row and matrix-multiply this with each consecutive 3 rows of the matrix, adding them at the end, so each time the matrix is doubled in length.

For an arbitrary 3-step sequence, we have to multiply these columns with the start sequence and do some bookkeeping which ultimately brings us to FS2:

```
   FS2=: 4 :'y ({.@[, (+/ .*)~)([, {: (+/ .*)"(1 _1) (#y)[\])^:x
(}.,+/)x:^:(64=3!:0 y) =/~ i. #y'          NB. wrap around!
```

Notice we use `x:^:(64=3!:0 y)` to guarantee the datatype of the input.

```
   3 FS2 1 2 3 4 5
```

```
1 2 3 4 5 15 29 56 109 214 423 831 1633
```

There is one problem however, the solution is slow and fat.

```
   (ts' 12 FS2 t') [ t=:>:i.5x
```

```
4.8792571 48033792
```

Compare this to

```
   (ts' 820 FS1 t')[t=:>:i.5x
```

```
0.05641251 9406848
```

with

```
   #L:0(820 FS1 t);12 FS2 t
```

```
+----+----+
|4100|4101|
+----+----+
```

which is 5 times as lean and 86 times as fast.

We can however speed up things a bit by the following improvements.

```
   FS2b=: 4 :'{:"1(, [:(](+/ .*)"2~ [:,/ (#y) +/\@:|.\ {."1@{:)
(#y)[\])^:x 0 0 0 0 1,.y'
```

```
   3 FS2b 1 2 3 4 5
```

```
1 2 3 4 5 15 29 56 109 214 423 831
```

with

```
   ts' 12 FS2b t'
```

```
1.94367 20040832
```

roughly twice as fast and twice as lean.

3.4.FS3

Well, as elegant and easy as F3 looks, the generalized version for *n*-step fibseqs FS3 is quite a bit more complicated. We have to construct the 'prefix of a prefix. Suppose we have the numbers `1 2 3 4 5` as the start of an 5-step sequence. Then

```
   +/\.1 2 3 4 5
```

```
15 14 12 9 5
```

delivers the first next number, 15. But after that we need 15+14=29, then 15+29+12=56, then 15+29+56+9=109 and finally 15+29+56+109+5=214.

The most easy way I found for this was

```
   ([:((++/),])/&.|. [:+/\. ])1 2 3 4 5
15 29 56 109 214
```

But then the generalization is not far off.

```
   FS3=: [:, ([:((++/),])/&.|. [:+/\. ])^: (<@[`])
```

with

```
   3 FS3 1 2 3 4 5
1 2 3 4 5 15 29 56 109 214 423 831 1633 3210 6311
```

So the left hand variable is the start sequence, the right hand is the number of iterations.

3.5. Performance comparison

Now we have all verbs generalized and the question is how they will perform. In this contest we will generate some 4000 items of an extended 5-step and a 500-step sequence subsequently and compare the performance. Notice that for non-extended sequences the performance figures can be much different.

```
    t=:>:i.5x
    #L:0(4095 FS0 t);(820 FS1 t);(12 FS2b t);820 FS3 t
+----+----+----+----+
|4100|4100|4101|4100|
+----+----+----+----+
   2-:/\ 4100{.L:0(4095 FS0 t);(820 FS1 t);(12 FS2b t);820 FS3 t
1 1 1
    datatype L:0(4095 FS0 t);(820 FS1 t);(12 FS2b t);820 FS3 t
+--------+--------+--------+--------+
|extended|extended|extended|extended|
+--------+--------+--------+--------+
```

This seems fair enough to do the comparison.

```
   [prf=:,.10 ts L:0'(4095 FS0 t)';'(820  FS1 t)';'(12 FS2b t)';
'820 FS3 t'     NB. wrap around!
```

```
+------------------+
|0.77643453 8013184 |
 +------------------+
|0.057555155 9405696|
+------------------+
|1.9468407 20040832 |
+------------------+
|0.024692076 8066176|
+------------------+
```

place	rlprf	rlt	rls	verbs
0	1.00	1.00	1.01	FS3
1	2.72	2.33	1.17	FS1
2	31.24	31.44	1.00	FS0
3	195.89	78.84	2.50	FS2b

Obviously FS3 is the most efficient as we can see with

```
('FS0';'FS1';'FS2b';'FS3') dspl rnkng >,prf
```

However if we enlarge t considerably

```
    t=:>:i.500x
```

we need to omit FS2b since it is (still) too slow.

```
   #L:0(4000 FS0 t);(9  FS1 t);9 FS3 t
+----+----+----+
|4500|4500|4500|
+----+----+----+
   [prf=: ts L:0'(4000 FS0 t)';'(9  FS1 t)';'9 FS3 t'
+---------------+-------------------+----------------+
|4.254658 7896832|36.337169 1.0148301e8|1.7803189 9897472|
+---------------+-------------------+----------------+
```

and with

```
('FS0';'FS1';'FS3') dspl rnkng >,prf
```

we get quite a different ranking. The overall winner is still FS3.

place	rlprf	rlt	rls	verbs
0	1.00	1.00	1.25	FS3
1	1.91	2.39	1.00	FS0
2	209.28	20.41	12.85	FS1

Addendum

The performance table is produced by the expression

```
  ('F0';'F1';'F2';'F3') dspl rnkng scores '2 F0^:(_2+2^12) 1
1x';'F1 2^11';'}:F2 12';'F3 2^11'     NB. wrap around!

+-----+------+------+----+-----+

|place|rlprf |rlt   |rls |verbs|

+-----+------+------+----+-----+

|0    |  1.00|  1.00|1.06| F3  |

+-----+------+------+----+-----+

|1    |  5.09|  3.30|1.63| F1  |

+-----+------+------+----+-----+

|2    | 36.98| 37.03|1.06| F2  |

+-----+------+------+----+-----+

|3    |112.61|119.46|1.00| F0  |

+-----+------+------+----+-----+
```

The verbs are defined by

```
scores=:  10 ts&> ]

rnkng=: 3 : '([:(,~/:@/:@{.) [:(%<./)"1(,~*/)) &.|: y'

dspl=: 4 : 0

z=: ',0.2,0.2,0.2'(8!:0) y

z=: ('place';'rlprf';'rlt';'rls'),z

 ({.,(/:{."1)@}.)('verbs';((<' '),&.>x)),.~z

)


   datatype           NB. wrap around!

3 : ((]`(<:@<:)@.(9<]))2^.(3!:0) y)>@{
'boolean';'literal';'integer';'floating';'complex';'boxed';
'extended';'rational';'sparse boolean';'sparse literal'; 'sparse
integer';'sparse floating';'sparse complex';    'sparse
boxed';'symbol';'unicode'
```

Notes on Equivalence Between some Boolean Arrays of Different Rank

Nollaig MacKenzie

3781 Panorama Crescent

Chemainus, BC, V0R 1K4

Canada

ABSTRACT

I consider some boolean arrays of shape N#2, serving as J-style truth-tables. A rank N array will, characteristically, correspond to an equivalence class of schemata in N letters. An array of rank N may equivalent to one of rank M, M>N. Some axes of the rank M array are superfluous, as are letters of the corresponding schemata.

I introduce two J functions, *padarr*, which adds superfluous axes and letters, and *unpad*, which goes in the opposite direction. The result of *unpad* I call a tbox. The function *tbfn* applies the boolean functions to tboxes.

I conclude with a remark about the number of truth-functions in N variables which cannot be reduced by *unpad*.

The function *truarr*, details left unspecified here, takes a schema such as '(a*.b)<:c' as argument and returns a J truth table, that is a boolean array whose axes match the alphabetical order of the letters in the schema.

Equivalence is sometimes defined: Two schemata are equivalent if they have the same truth-table. This needs qualification; consider:

```
   truarr &.> 'a';'a+.a*.b'
┌─────┬─────┐
│0 1  │0 0  │
│     │1 1  │
└─────┴─────┘
```

The tables are different, though they “say the same”; a computation leading from the first to the second would be logically vacuous, in the manner of +./&0 0. The function *padarr*

```
      /:@[ |: (#@[ - #@$@]) (0 0 +./~ ])^:[ ]
```

will perform such computations, inserting new axes as the letters in the left argument specify:

```
   ('bac' padarr 0 1) ; 'abc' padarr 0 1
┌─────┬─────┐
│0 0  │0 0  │
│1 1  │0 0  │
│     │     │
│0 0  │1 1  │
│1 1  │1 1  │
└─────┴─────┘
```

An imperfect inverse of *padarr* is *unpad*:

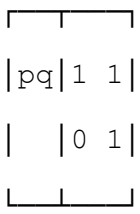
```
3 : 0
((#@$ y){. alpha) unpad y NB. alpha is the lower case alphabet.
:
nn=. i. 0
r=. #@$ y
for_cc. i. r do. nn=. nn, (({."1  -: {:"1) cc |: y) # cc end.
sv=. (<0) nn } r # <0 1
cv=. 0 nn } r # 1
(cv # x); (< sv){ y
)
```

The left argument of unpad is a string of letters, one for each axis, the right a boolean array, the result two boxes, one containing a string of letters, the other a boolean array. These are the *operative* letters and axes.

```
   'abc' unpad 2 2 2$0 0 1 1 0 0 1 1
┌──┬─────┐
│b │0 1  │
└──┴─────┘
```

We can say that a table of N+M axes is equivalent to a table of N axes if the N+M table can be got from the smaller by way of *padarr*. *unpad* will reveal, for any table, whether there is such a smaller.

Call an object of the form (***letters;boolean_array***) a tbox. It contains the same information as do various boolean schemata. For example, the tbox



contains the same information as the schema 'p<:q' and its equivalents ('(-.p)+.q', etc.).

To apply the boolean functions to tboxes we need an adverb, say *tbfn*:

```
1 : 0

'xr yr'=. y

xr unpad u yr

:

'xl yl xr yr'=. x,y

N=. #ov=. xl ([ #~ [ e. ]) xr

ll=. (xl-.ov),(xr-.ov),ov

I=. i.&ov@[|:]

ll unpad (xl I yl) (u"N"(N,_)) xr I yr

)
```

If the left and right argument tboxes have identical letter strings, *tbfn* simply applies ***u*** to the left and right arrays; if they have completely different letter strings, it applies a ***u*** outer product to the arrays. If there is an N-letter overlap *tbfn* transposes those N axes to the end, then does a ***u*** rank N outer product. (This can be written either '***u***”N/' or '***u***”N”(N,_)', but Dan Bron has convinced me that the latter is more illuminating).

For an alphabetized tbox, use *tbfna*, which differs from *tbfn* thus:

```
          ....
v=. (i. /:~) ll
(v{ll) unpad v|:(xl I yl) (u"N"(N,_)) xr I yr
          ....
```

For example:

```
   ('ab';2 2$1 0 0 0) ((+. tbfn) (<@[,<@]) (+. tbfna)) 'ac';2
2$0 0 0 1
```

```
┌─────────┬─────────┐
│┌───┬───┐│┌───┬───┐│
│|bca|1 0|||abc|1 1||
||   |1 1|||   |0 0||
||   |   |||   |   ||
||   |0 0|||   |0 1||
||   |0 1|||   |0 1||
│└───┴───┘│└───┴───┘│
└─────────┴─────────┘
```

Truth-tables are often used to test for implication, equivalence, contradiction, and the like, and tboxes can also do such work. For example, to check that 'a<:b' and 'a' imply 'b', that is, that '((a<:b)*.a)<:b' is a tautology, evaluate:

```
   (('ab';2 2$1 1 0 1) (*. tbfn) 'a';0 1) (<: tbfn) 'b';0 1
```

```
┌┬┐
||1|
```

```
┌┴┐
└─┘
```

Tboxes wear their rank on their faces. A tautology has no axis; if a tautologous truth-table has axes, they should be seen as introduced implictly by something like *padarr*. And likewise for higher ranks. When we say, for example, that there are only ten dyadic truth functions,

```
┌─┬─┬─┬─┬─┬─┬─┬─┬─┬─┐
│*.│>│<│+:│+.│>:│<:│*:│=│~:│
└─┴─┴─┴─┴─┴─┴─┴─┴─┴─┘
```

though there are 16 possible 2-axis boolean tables (2 2#:4 4#:i.16), we can be understood as saying that these tables come from these phrases:

```
┌─────────┬─────────┬─────────┬─────────┬─────────┬─────────┐
                        ┌──┐
  │0 0       │0 0       │0 1       │1 1       │1 0       │1 1
                        │
  │0 0       │1 1       │0 1       │0 0       │1 0       │1 1
                        │
├─────────┼─────────┼─────────┼─────────┼─────────┼─────────┤
                        └──┤
    │'ab'&padarr 0│'ab'&padarr 0 1│'ba'&padarr 0 1│'ab'&padarr 1 0│'ba'&padarr 1
                                                    0│'ab'&padarr 1│
└─────────┴─────────┴─────────┴─────────┴─────────┴─────────┘
                        └──┘
```

There are 2 niladic functions, and 1 way (0!2) of placing the axes; 2 monadic functions, and 2 ways (1!2) of placing the “padding” axis. Generalize, and we have *numfns*, which calculates the number of truly N-adic functions among the 2^2^N tables of N axes:

```
3 : 0
fnsbelow=. 0
   for_C. i. y do. fnsbelow=. fnsbelow + (C!y) * numfns C end.
(2^2^y) - fnsbelow
```

)

A correctness proof for *numfns*: I googled the series 2, 2, 10, 218, 64594, 4294642034, 18446744047940725978 .... It is, in fact, A000371 in *The On-line Encyclopedia of Intege*

Using Color to See Patterns in Numeric Tables

Linda Alvord

lindaalvord@verizon.net

This paper uses J to change numeric tables into color images. The images clarify important aspects of the data and instantly yield a wealth of information.

In English, "Rover **catch** ball" is a sentence. "**Drop** ball" is a command. It implies a subject will do the dropping. The verbs catch and drop have results. In J, the verb in a sentence is dyad. The verb in a command is a monad. Both have results.

In art a collection of paint colors is a palette. In J, colors are made with three column tables of numbers. Each row represents a unique color. In mathematics, a numeric table is a matrix. The J sentence "PAL **viewmat** DATA" requires a color palette, PAL. The name stands for a table of colors. The dyadic verb viewmat, for view the matrix, requires a second numeric table, DATA.

The result is a color image. The magnitude of each number in the data corresponds to a color in the result. The image matches the brightest colors with the largest numbers. The range of colors provides an overview of the highs and lows of the data.

A pixel is the smallest unit of a color image. Each pixel is made from red, green and blue, or RGB light. A 3-item list like 0 0 0 defines black, or no light. Each light component may range in an integer value from 0 to 255. To produce white, all lights are at full intensity or 255 255 255. A list like 43 236 174 is a recipe for one of the 256 * 256 * 256 graphic colors. The product */256 256 256 is 16777216 revealing that it is possible to have almost 17 million colors.

One of the ways to use J is to have a dialog. To capture the conversation, the left column below shows the J entry indented. At the right are notes from the reader to further clarify the conversation. EEach new J verb is named in bold face type followed by the actual symbol or symbols

The length of the hypotenuse of a right triangle is the square root of the sum of the squares of the measures of the lengths of each of the legs of a right triangle. Pythagoras might have done this.

```
  *:3 4                                    square *:
each item in list 3 4
9 16
   +/9 16                                  plus insert
+/ between items
25                                         is a J dyad
   %:25                                    Square root
%: of each item
5
```

```
   %:+/*:3 4                                          Result from a
single entry
5
```

The next section develops three palettes. RGB has 8-colors including black, blue, green, cyan, red, magenta, yellow and white. Unlike paint, red and green light makes an even brighter color yellow. Red and blue is magenta light and green and blue light is cyan in color. PAL, a 6-color palette eliminates black and white. These are the common hues on a color wheel. GRB is an 8-color palette which interchanges the red and green columns. This is the palette which will be the focus of the remainder of the paper.

The left argument of viewmat is a numeric table of colors. The following steps create the palettes.

```
    load 'viewmat'                                    load
'filename' for viewmat verb

   i.8                                                integers
i. from 0 to 7
0 1 2 3 4 5 6 7

   #:i.8                                              antibase 2
#: converts the
0 0 0                                                 8  numbers
from 0 to 7
0 0 1                                                 into 8
numbers in base 2
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0                                                   The rows
are from 0
1 1 1                                                   to 7 in
the binary system


  ]RGB=:( #:i.8){0 255                                 RGB is =:
the binary table
                                                       The data
now has a name
  0   0   0                                           same ]
displays the table RGB
  0   0 255                                           The name and
the table agree
  0 255   0
  0 255 255
255   0   0
255   0 255
255 255   0
255 255 255


   ]PAL=: _1 }.  }. RGB                               behead }.
the first row
  0   0 255                                            eliminates
it
  0 255   0
  0 255 255                                             _1 drop
}. the last row
255   0   0                                           This is a
dyad and the left noun
255   0 255                                           indicate
which rows to drop.
255 255   0                                          _1 is the
last row

                                                         Here's the
magical ingredient!

   ]GRB=:1 0 2 {"1 RGB
  0   0   0                                            1 0 2 from
rank 1  {"1 RGB
```

```
  0   0 255                                                    Rank 0 of
a table means rows
255   0   0                                                    Rank 1 of
a table means columns
255   0 255                                                    1 0 2
selects columns two,
  0 255   0                                                    one and
then three from RGB
  0 255 255                                                    The
brightest light is now
255 255   0                                                    in first
place.
255 255 255
```

```
   'yellow';'white'                                            'yellow'
link ; 'white'
┌──────┬─────┐                                                Links
together the two
│yellow│white│                                                boxed
nouns
└──────┴─────┘
```

```
 'black';'blue';'green';'cyan';'red';'magenta';'yellow';'white'
```

black	blue	green	cyan	red	magenta	yellow	white

```
   >'black';'blue';'green';'cyan';'red';'magenta';'yellow';'white'
Black
blue                                                           open >
each box and                                        green
form rows of a table.
cyan                                                           All rows
are the size
red                                                            of the
largest row.
magenta
yellow
white
```

```
   GRB;>'black';'blue';'green';'cyan';'red';'magenta';'yellow';'white'
┌─────────────┬─────────┐
│  0   0   0  │black    │                                       Here
link joins a numeric
│255   0   0  │green    │                                       table
to a literal table.
│255   0 255  │cyan     │
│  0 255   0  │red      │
│  0 255 255  │magenta  │
│255 255   0  │yellow   │
│255 255 255  │white    │
└─────────────┴─────────┘
```

```
A=:RGB;>'black';'blue';'green';'cyan';'red';'magenta';'yellow';'white'
   B=:PAL;>'blue';'green';'cyan';'red';'magenta';'yellow'
   C=:GRB;>'black';'blue';'red';'magenta';'green';'cyan';'yellow';'whi
te'
```

```
 A;B;C                                                         A summary of
three palettes
┌ ──────────────────────────┬──────────────────────┬─────────┐
│┌──────────┬──────┐ ┌──────┬──────┐ │  0   0   0│black    │
```

0	0	0	black
0	0	255	blue
0	255	0	green
0	255	255	cyan
255	0	0	red
255	0	255	magenta
255	255	0	yellow
255	255	255	white

0	0	255	blue
0	255	0	green
0	255	255	cyan
255	0	0	red
255	0	255	magenta
255	255	0	yellow

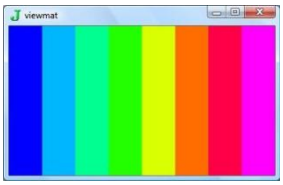
0	0	255	blue
255	0	0	red
255	0	255	magenta
0	255	0	green
0	255	255	cyan
255	255	0	yellow
255	255	255	white

Now we have the palettes viewmat requires to make images.

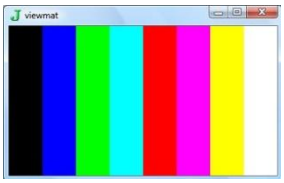
For data, start with i.8 a simple list. The default left noun of viewmat is a 6-color palette of blue, cyan, green, yellow, red and magenta. Once you load 'viewmat' you can enter a command. The image appears. It has 8 colors from the 6-color default palette. The colors do not include orange but it appears between yellow and red. to The second image on the right uses the 8-color RGB palette. Both images have no title.

```
   load 'viewmat'
```

viewmat i.8



RGB viewmat i.8

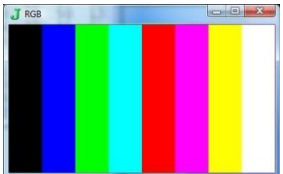


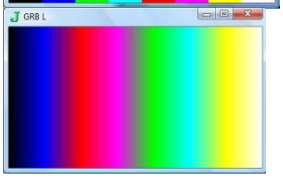
It is possible to put a title with the image. First use link to box the data and box the caption. You must enclose i.8 in parentheses. The four images below are the result of entering the four sentences. Note that the image is now in a separate wrapper. It can be copied and pasted into pait, saved as a jpg image and the put in a document. At some point, the image without the wrapper could appear right in the dialog.

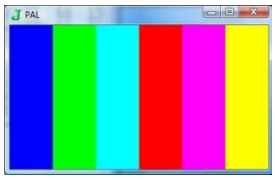
```
   (i.8);'RGB'
```

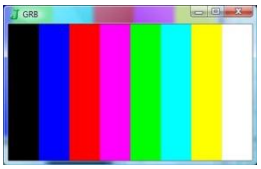
0	1	2	3	4	5	6	7	RGB

```
RGB viewmat (i.8);'RGB'                                                  First
link (i.8); 'RGB'
PAL viewmat (i.6);'PAL'                                                  viewmat
uses the data in the
GRB viewmat (i.8);'GRB'                                                  second
box as a caption for
GRB viewmat (i.1000);'GRB L'                                             the
image'
                                                                         RGB
viewmat (i.8) makes image.
```









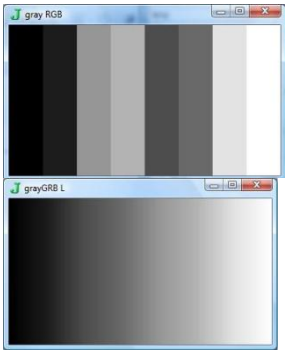
The last image illustrates that it is just as easy to have 1000 colors as it is to have 8.

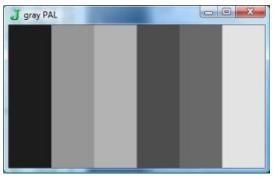
This section transforms color images into grayscale images. The use of the list of weights came from an article in Wikipedia. An address is
http://en.wikipedia.org/wiki/Grayscale

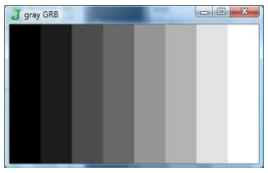
```
      0.3 0.59 0.11*"1 PAL                                                0.3
0.59 0.11 times *"1
   0      0 28.05                                                         PAL
   0 150.45     0                                                        0.3
0.59 0.11 times each
   0 150.45 28.05                                                       row
of PAL weights
76.5      0     0                                                       the
colors to
76.5      0 28.05
equalize intensity.
76.5 150.45     0

   +/"1[ 0.3 0.59 0.11*"1 PAL                                           Sum
of each row row
28.05 150.45 178.5 76.5 104.55 226.95

    0.5+"0  +/"1[ 0.3 0.59 0.11*"1 PAL
0.5 + sum of each row28.55 150.95 179 77 105.05 227.45

   <. 0.5+"0  +/"1[ 0.3 0.59 0.11*"1 PAL
floor <. Rounds
28 150 179 77 105 227                                                   to
smaller integers

  3#"0<. 0.5+"0  +/"1[ 0.3 0.59 0.11*"1 PAL                             3
copies"0 of
 28  28  28                                                             each
item in the result.
150 150 150                                                             A
dyad, 3 indicate ho many.
179 179 179
However, the list
 77  77  77                                                             28
150 179 77 105 227
105 105 105                                                             is
not in order.
227 227 227

  gray=: 13 :'3#"0 +<.0.5+"0 +/"1]0.3 0.59 0.11*"1 y'                   This
creates the verb gray

  gray RGB
  0   0   0
gray RGB
 28  28  28                                                             load
is another verb
150 150 150                                                             with
a name instead of 179 179 179
symbols.
 77  77  77
Verbs are usually named
105 105 105
with small letter and
227 227 227
nouns with capitals.
255 255 255

     gray GRB                                                           gray
GRB is an improvement
  0   0   0
 28  28  28                                                             Now
the gray colors range
 77  77  77                                                             from
a dark to light
105 105 105
Finally the values are in
150 150 150
numeric order.
179 179 179
227 227 227
255 255 255
```

These sentences create the 4 grayscale images shown below.

```
(gray RGB) viewmat (i.8);'gray RGB'
(gray PAL) viewmat (i.6);'gray PAL'
(gray GRB) viewmat (i.8);'gray GRB'
(gray GRB) viewmat (i.1000);'grayGRB L'
```







It is apparent that GRB and gray GRB are the best palettes to shade from dark to light.

The section which follows looks at possible tables of numeric data. You just saw eight images about numeric data. Here are more.

```
  i.5                                                    Integers
from 0 to 4
0 1 2 3 4

   i._5                                                  Reverse the
order of i.5
4 3 2 1 0

   i:5                                                   steps i:
of integers
_5 _4 _3 _2 _1 0 1 2 3 4 5                               _5 to 5

   i:_5                                                  Reverse the
order of i:5
5 4 3 2 1 0 _1 _2 _3 _4 _5

  i:"0 i:2                                               steps rank
0 i:"0 i:2
 2   1   0  _1 _2
 1   0  _1   0  0
 0   0   0   0  0                                       apply i: to
each item in i:2 or
_1   0   1   0  0                                       _2 _1  0  1  2
_2  _1   0   1  2

   i:0                                                   Integers in
steps from
0                                                        _0 to 0 is
only 0

    i.0                                                  Integers
counting to 0
                                                         or nothing
is empty

   ]D=:i:"0 i:2
 2   1   0  _1 _2
 1   0  _1   0  0
 0   0   0   0  0
_1   0   1   0  0
_2  _1   0   1  2

   ,D                                                    ravel ,
makes a
```

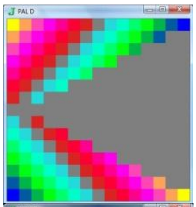
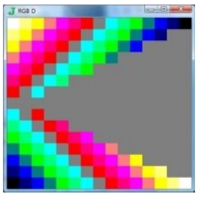
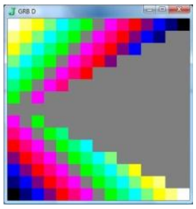
```
2 1 0 _1 _2 1 0 _1 0 0 0 0 0 0 0 _1 0 1 0 0 _2 _1 0 1 2                    list of
items in D

   $,D                                                                    shape Of
$ of list
25                                                                        counts the
items

   +/,D                                                                   plus
between items of ,D
0                                                                         is the sum
of list D

   LD=:i:"0 i:500                                                         LD is a
large list of table= D

  $LD                                                                     1001 rowS
and
1001 1001                                                                 1001
columns

   $,LD                                                                   Size of
list in D.
1002001

  1001*1001                                                               1001 times
* 1001
1002001                                                                   A dyadic
verb
```

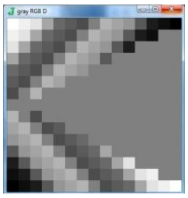
D is now a a larger table of values. Look for the largest and smallest items. Remember that the largest numbers
select white and smallest numbers are black. The size of numbers in tables is relative to the entire collection.

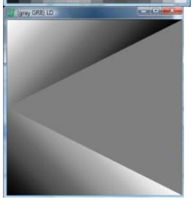
```
   ]D=:i:"0 i:7
 7  6  5  4  3  2  1  0 _1 _2 _3 _4 _5 _6 _7
 6  5  4  3  2  1  0 _1 _2 _3 _4 _5 _6  0  0
 5  4  3  2  1  0 _1 _2 _3 _4 _5  0  0  0  0
 4  3  2  1  0 _1 _2 _3 _4  0  0  0  0  0  0
 3  2  1  0 _1 _2 _3  0  0  0  0  0  0  0  0
 2  1  0 _1 _2  0  0  0  0  0  0  0  0  0  0
 1  0 _1  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
_1  0  1  0  0  0  0  0  0  0  0  0  0  0  0
_2 _1  0  1  2  0  0  0  0  0  0  0  0  0  0
_3 _2 _1  0  1  2  3  0  0  0  0  0  0  0  0
_4 _3 _2 _1  0  1  2  3  4  0  0  0  0  0  0
_5 _4 _3 _2 _1  0  1  2  3  4  5  0  0  0  0
_6 _5 _4 _3 _2 _1  0  1  2  3  4  5  6  0  0
_7 _6 _5 _4 _3 _2 _1  0  1  2  3  4  5  6  7
```

```
]D=:i:"0 i:7
LD=:i:"0 i:1000
RGB viewmat D;'PAL D'
PAL viewmat D;'RGB D'
GRB viewmat D;'GRB D'
GRB viewmat LD;'GRB LD'
(gray PAL)viewmat D;'gray PAL D'
(gray RGB)viewmat D;'gray RGB D'
(gray GRB)viewmat D;'gray GRB D'
(gray GRB)viewmat LD;'gray GRB LD'
```

Here are 8 images. The four made with GRB or gray GRB have good gradients from black to white.

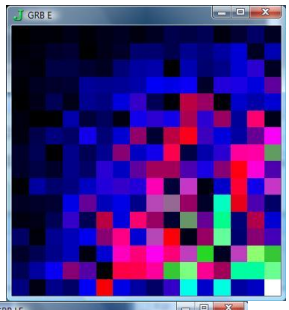


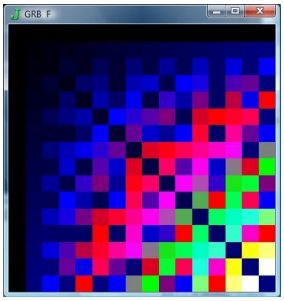
The connection between the data table and the colors is significant. It is apparent from the corners where the large and small items are located.

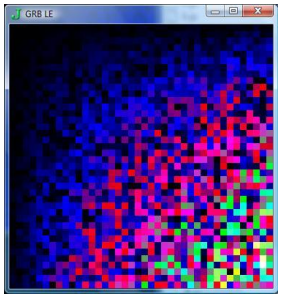
Next look at the two tables E and F. It would take a considerable effort to understand these two tables. There are similarities and yet there are differences.

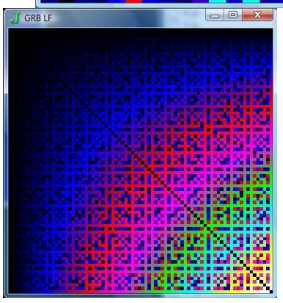
```
    E;F
┌──────────────────────────────────────────────────────┬────────────────────────────────┐
┌──────────────────────┐
│ 0  1  1  1  2  2  0  5  0   3   4   6   9  13   4  10│0  0  0  0  0  0  0   0   0   0
 0   0   0   0   0   0│
│ 0  0  4  0  9  4  3  9 10   0  17   6  17  21  23   4│0  1  2  3  4  5  6   7   8   9
10  11  12  13  14  15│
│ 2  2  1 11 13 17 18 17 13  16  24  26  38   9  22  15│0  2  2  6  4 10  6  14   8  18
10  22  12  26  14  30│
│ 3  6  5  1  4 14  5 14  8  27   9  44  23  53  27  18│0  3  6  3 12 15  6  21  24   9
30  33  12  39  42  15│
│ 3  5  0  0 23 17  0  2  3  26  30  23   0  27  25  28│0  4  4 12  4 20 12  28   8  36
20  44  12  52  28  60│
│ 5  8 16 14  3 14 17 28 39  19  53  12  13  71  87  41│0  5 10 15 20  5 30  35  40  45
10  55  60  65  70  15│
│ 1  9  3 26 28  4  4 42 28   1  54  26  89  66  38  95│0  6  6  6 12 30  6  42  24  18
30  66  12  78  42  30│
│ 0  4 15 18 12  3  3 21 43  66  32  17  23  84  47  40│0  7 14 21 28 35 42   7  56  63
70  77  84  91  14 105│
│ 1  4  8  3  6 11 52 55 43  23  34  53  87  31 126  51│0  8  8 24  8 40 24  56   8  72
40  88  24 104  56 120│
│ 2  0 10 10 36 29 46 27 66  70  20  31  21   8  93 150│0  9 18  9 36 45 18  63  72   9
90  99  36 117 126  45│
│ 8  8 25  9 11 44 52  3 83  80  65 127  98  24 124  63│0 10 10 30 20 10 30  70  40  90
10 110  60 130  70  30│
│ 1 10 12 28 12 51 26 50 62  54   8 106   8  56  26 148│0 11 22 33 44 55 66  77  88  99
110  11 132 143 154 165│
│ 5  3  7 43 27 11 48 17 77  17  49  33 162 127  85  81│0 12 12 12 12 60 12  84  24  36
60 132  12 156  84  60│
│ 5  0 29 12 63 54 25 14  1 129  69  43 161  53  35  41│0 13 26 39 52 65 78  91 104 117
130 143 156  13 182 195│
│ 3  9 27  0  0  2 25 28 26   9 101  46 111 140  83 228│0 14 14 42 28 70 42  14  56 126
70 154  84 182  14 210│
│15 30 25 29 44 90 68 32 41  20  93 111 190 116 198  57│0 15 30 15 60 15 30 105 120  45
30 165  60 195 210  15│
```

Images. E has no apparent pattern, but F does. Look for ? as a dyad verb called deal or a monad ? called roll. These are J verbs that produce random results. Both E and LE are images of random data. Look at the code at the end of this section. Numeric tables generated with J verbs typically have patterns. F and LF are created from similar tables made with a dyadic J verb. The code below shows the verb.









After observing these tables and images, look below to see how they are created.

```
   >:i.3
1 2 3
   (>:i.3)*/>:i.3                                                    times
table */ (>:i.3)*/>:i.3
1 2 3                                                                A
dyadic verb and / makes a
2 4 6
conventional table using the
3 6 9                                                             chosen
verb

   */~>:i.3                                                          times
table reflexive */~
1 2 3                                                             is a
monad that uses the right 2 4 6
noun as the left noun also.
3 6 9
Reflexive applies to any dyad.

   ?*/~>:i.3                                                        roll
? selects an integer
0 1 1                                                             from
i.n in each position
1 2 2                                                             in the
table
0 5 0

E=:?*/~>:i.16
GRB viewmat E;'GRB E'
F=:*./~ i.16                                                      (i.16)
LCM *. i.16 is the GRB  viewmat F;'GRB F'
least common multiple of pairs
LE=:?*/~>:i.40                                                    70 is
the LCM of 35 and 14
GRB viewmat LE;'GRB LE'
LF=:*./~ i.40
GRB viewmat LF;'GRB LF'
```

The verb viewmat is the work of Cliff Reiter found in “Fractals Visualization and J”. His verb, viewmat, produces the first image below with a default 6-color palette. RGB based on binary counting has black and white to form an 8-color palette. PAL removes black and white from RGB and is slightly different order of colors than the other 6-color palette. The fourth and fifth are GRB and gray GRB palaettes.

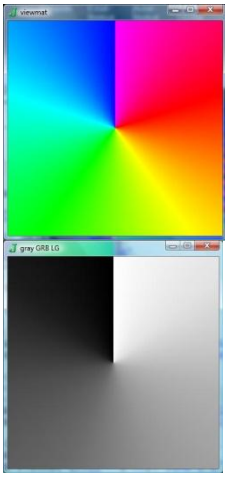
The verb at2 which he ilcludes in his text is interesting. It is beyond the scope of this article to explain it. However, even a small sample of its behavior is intreguing.

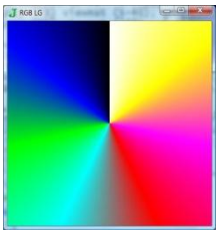
```
     at2=: 13 :'([:{:"1 *.) j./"1  i:4
```

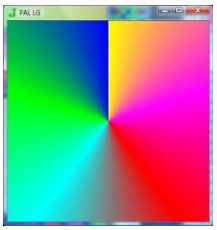
```
    ]SEE=:at2"0 /~ i:4
 _2.35619  _2.49809  _2.67795  _2.89661 3.14159  2.89661  2.67795  2.49809  2.35619
  _2.2143  _2.35619  _2.55359  _2.81984 3.14159  2.81984  2.55359  2.35619   2.2143
 _2.03444   _2.1588  _2.35619  _2.67795 3.14159  2.67795  2.35619   2.1588  2.03444
 _1.81577  _1.89255  _2.03444  _2.35619 3.14159  2.35619  2.03444  1.89255  1.81577
  _1.5708   _1.5708   _1.5708   _1.5708       0   1.5708   1.5708   1.5708   1.5708
 _1.32582  _1.24905  _1.10715 _0.785398       0 0.785398  1.10715  1.24905  1.32582
 _1.10715 _0.982794 _0.785398 _0.463648       0 0.463648 0.785398 0.982794  1.10715
_0.927295 _0.785398 _0.588003 _0.321751       0 0.321751 0.588003 0.785398 0.927295
_0.785398 _0.643501 _0.463648 _0.244979       0 0.244979 0.463648 0.643501 0.785398
```

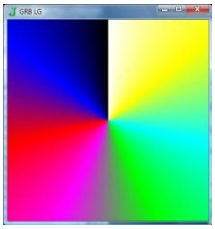
```
LG=:at2"0 /~ i:1000
viewmat LG
RGB viewmat LG;'RGB LG'
PAL viewmat LG;'PAL LG'
GRB viewmat LG;'GRB LG
(gray GRB)viewmat LG;'gray GRB LG'
```

The gray GRB image could be the face of a clock and in a sweep from 12 am. to 12 pm. The gradient is from white to black. GRB is similar sweepimg from white, through green , red and then blue and finally black.









Here’s the “Real Deal”. Real data is easy to understand with a color image. Grab a table and view it with GRB! Tama Traberman wrote a book called APL IN SOCIAL STUDIES, Published by I-APL around 1985. No doubt the information has changed since then. Here is a basic translation from APL. The table provides the population, area (probably square miles) and population density of each continent.

```
CONT=:>'EUROPE';' ASIA';' NORTH AMERICA';' SOUTH AMERICA';' AFRICA';' ANTARCTICA';' AUST
RALIA'
POP=:702300000 2896700000 400000000 271000000 551000000 1000 115800000x
AREA=:4017000 16990000 9366000 6881000 11688000 5100000 2966000x
DEN=:175 170 43 39 47 0 5
```

```
   ]CONT;DATA=:POP,.AREA,.DEN
,.  AREA,.DEN
┌─────────────┬──────────────────────┐
of equal length 7 form
│EUROPE       │ 702300000  4017000 175│
of shape 7 2.
│ ASIA        │2896700000 16990000 170│
│ NORTH AMERICA│ 400000000  9366000  43│
│ SOUTH AMERICA│ 271000000  6881000  39│
```

stitch

Two lists

a table

```
┌──────────────┬──────────────────────────┐
│ AFRICA       │ 551000000 11688000  47 │
│ ANTARCTICA   │      1000  5100000   0 │
│ AUSTRALIA    │ 115800000  2966000   5 │
└──────────────┴──────────────────────────┘
```

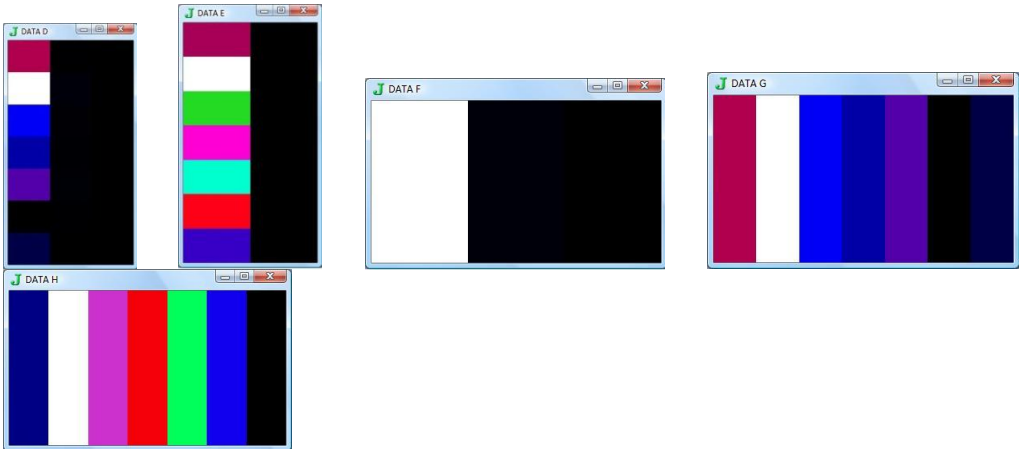
```
   DATA
 702300000  4017000 175
2896700000 16990000 170
 400000000  9366000  43
 271000000  6881000  39
 551000000 11688000  47
      1000  5100000   0
 115800000  2966000   5
    $CONT;DATA=:POP,.AREA,.DEN                                    Shape of
list with 2 items.
2
   $DATA                                                          Shape of
table with 7 rows
7 3                                                               3
columns.
```

```
GRB viewmat DATA;'DATA D'
GRB viewmat (1 2{"1 DATA);'DATA E'
GRB viewmat (1{DATA);'DATA F'
GRB viewmat (0{"1 DATA);'DATA G'
GRB viewmat (1{"1 DATA);'DATA H'

]D1=:5e6<DATA
]D2=:DATA<7e6
D1*.D2
GRB viewmat D1
GRB viewmat D1;'D1'
GRB viewmat D2;'D2'
GRB viewmat (D1*.D2);'D1 *. D2'
```

The first image of the data has the largest numbers in the first column. In the second image, columns two and three are selected. A large number is in the second column of the data also. This large number is only relative to those two columns. Remember, also, that the columns in J are numbered 0 1 2.

The third image is the third row and there is a high and two low numbers in that row. Next as explained earlier, the fourth image is of the first column. It has one very high number and all the other numbers are relatively small. These dark colors indicate the data is skewed torwrd small numbers. The final image shows that the AREA column has a more normal distribution.

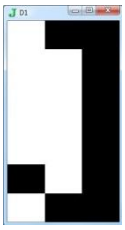
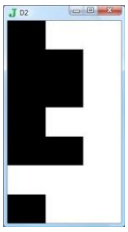
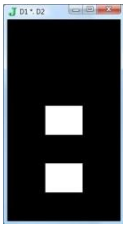
```
GRB viewmat DATA;'DATA D'
GRB viewmat (1 2{"1 DATA);'DATA E'
GRB viewmat (1{DATA);'DATA F'
GRB viewmat (0{"1 DATA);'DATA G'
GRB viewmat (1{"1 DATA);'DATA H'
```



To see the location of numbers between 5 and 7 million, first find all the DATA greater than 5 million. Next find the numbers less than 7 million. In scientific notation 5e6 means 5 and 6 zeros or 5000000. In the tables below, zero is a false and 1 is true so true is white. Now find the positions where both are true.

```
]D1=:5e6<DATA                                                    less
than < 5e6<DATA
1 0 0                                                            compared
5000000 to each item
1 1 0                                                            in data.
Each result is true or
1 1 0                                                            false.
1 1 0
1 1 0                                                            If 1 is
true and 0 false, the
0 1 0                                                            they can
also be numbers and
1 0 0                                                            used with
+, * and other verbs.

   ]D2=:DATA<7e6
0 1 1
0 0 1
0 0 1
0 1 1
0 0 1
1 1 1
0 1 1

   D1*.D2.                                                       LCM
(AND) *.  1 *. 1 is 1
0 0 0
0 0 0                                                            The LCM
of 1 and 0 is 0
0 0 0                                                            The LCM
of 0 and 0 is 0
0 1 0
0 0 0
0 1 0                                                            If (true
and true) or (1 and 1)
0 0 0                                                            or (1 *.
1) the answer is true.


GRB viewmat D1;'D1'
GRB viewmat D2;'D2'
GRB viewmat (D1*.D2);'D1 *. D2'
```

See how these images look.

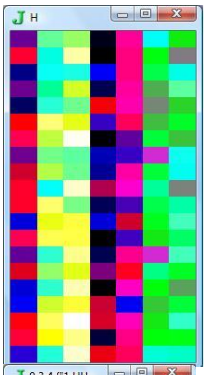




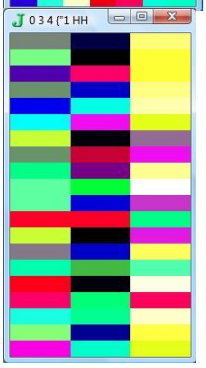
Simulating data has some potential pitfalls which are illustrated next. First build a table ALL with random data in each column. Then make tables that reorder the columns each time ALL is run. Create tables H, I, J and K. Also, create HH, II, JJ and KK to capture a copy of the random tables. You need them to see images of the random tables.

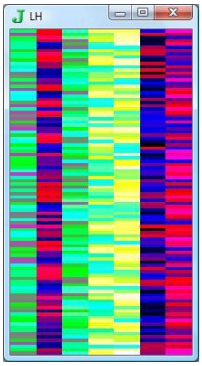
The sentences that apply to the remaining portion of the paper are at the end. The entire section can be moved to an ijs file. Whenever you run it in a new session of J you should get images that look like the ones that follow. However if you run the code again you will get different images. An advantage is that you get new tables to consider. Sometimes the results are puzzling. It may also help you to design data that many be appropriate in your own area of interest and then see if color images are helpful.

After each table you see four images. The first are images of the entire table. The second images are made using the same verbs and appling them to long lists. The shape is 7 1100 instead of 20 7. The third and fourth images in illustrate different ways to view the tables.

133	321	331	71	189	301	252
170	297	381	66	184	258	231
91	304	307	112	184	269	298
134	283	342	81	190	240	320
84	310	323	159	193	233	247
161	372	345	124	178	243	263
176	338	395	67	131	266	244
134	324	320	100	124	216	305
152	336	323	94	191	266	300
168	303	387	146	199	284	232
168	370	325	82	127	269	286
107	346	362	107	152	261	315
176	365	363	66	125	251	274
132	311	376	81	187	216	315
155	330	345	137	167	280	262
106	311	383	67	197	257	238
112	341	360	151	111	245	266
163	368	397	153	195	251	312
171	350	375	77	185	257	259
121	309	388	157	178	295	292





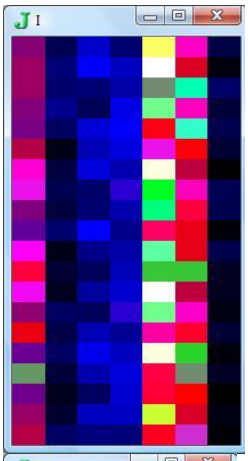


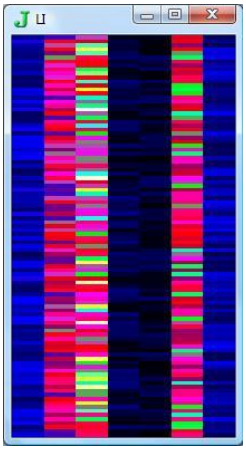
Further investigation of columns 2 and 3 show where the largest values re located. Columns 1, 4 and 5 do the same for the three columns with the smallest numbers. These new images show relative values of those two subsets.

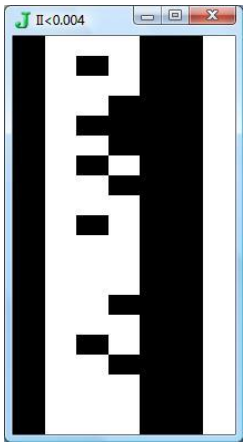
II

0.00529101	0.00311526	0.00396825	0.00332226	0.0140845	0.0075188	0.00302115
0.00543478	0.003367	0.004329	0.00387597	0.0151515	0.00588235	0.00262467
0.00543478	0.00328947	0.0033557	0.00371747	0.00892857	0.010989	0.00325733
0.00526316	0.00353357	0.003125	0.00416667	0.0123457	0.00746269	0.00292398
0.00518135	0.00322581	0.00404858	0.00429185	0.00628931	0.0119048	0.00309598
0.00561798	0.00268817	0.00380228	0.00411523	0.00806452	0.00621118	0.00289855
0.00763359	0.00295858	0.00409836	0.0037594	0.0149254	0.00568182	0.00253165
0.00806452	0.00308642	0.00327869	0.00462963	0.01	0.00746269	0.003125
0.0052356	0.00297619	0.00333333	0.0037594	0.0106383	0.00657895	0.00309598
0.00502513	0.00330033	0.00431034	0.00352113	0.00684932	0.00595238	0.00258398
0.00787402	0.0027027	0.0034965	0.00371747	0.0121951	0.00595238	0.00307692
0.00657895	0.00289017	0.0031746	0.00383142	0.00934579	0.00934579	0.00276243

0.008	0.00273973	0.00364964	0.00398406	0.0151515	0.00568182	0.00275482
0.00534759	0.00321543	0.0031746	0.00462963	0.0123457	0.00757576	0.00265957
0.00598802	0.0030303	0.00381679	0.00357143	0.00729927	0.00645161	0.00289855
0.00507614	0.00321543	0.00420168	0.00389105	0.0149254	0.00943396	0.00261097
0.00900901	0.00293255	0.0037594	0.00408163	0.00662252	0.00892857	0.00277778
0.00512821	0.00271739	0.00320513	0.00398406	0.00653595	0.00613497	0.00251889
0.00540541	0.00285714	0.003861	0.00389105	0.012987	0.00584795	0.00266667
0.00561798	0.00323625	0.00342466	0.00338983	0.00636943	0.00826446	0.00257732







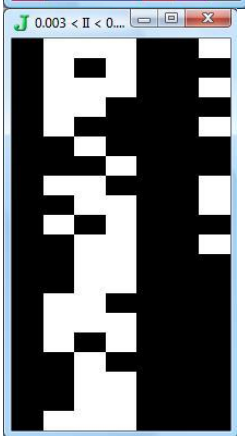
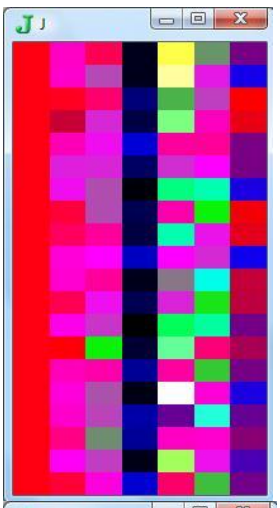


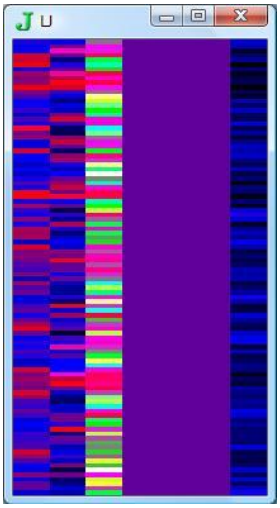
Image three shows where II is less than 0.004. Image four shows only those numbers between 0.003 and 0.004.

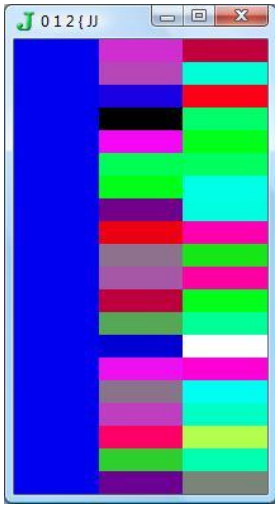
JJ

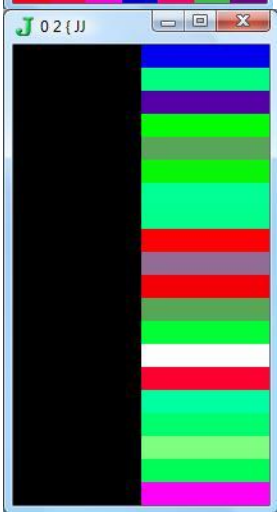
1	1.27381	1.09967	0.221184	2.66197	1.59259	0.761329
1	1.28571	1.47674	0.222222	2.78788	1.40217	0.606299
1	1.02013	1.14126	0.368421	1.64286	1.46196	0.970684
1	0.884375	1.425	0.286219	2.34568	1.26316	0.935673
1	1.25506	1.38627	0.512903	1.21384	1.20725	0.764706
1	1.41445	1.41975	0.333333	1.43548	1.36517	0.762319
1	1.38525	1.48496	0.198225	1.95522	2.03053	0.617722
1	1.0623	1.48148	0.308642	1.24	1.74194	0.953125
1	1.12	1.21429	0.279762	2.03191	1.39267	0.928793
1	1.30603	1.36268	0.481848	1.36301	1.42714	0.599483
1	1.29371	1.20818	0.221622	1.54878	2.11811	0.88
1	1.09841	1.38697	0.309249	1.42056	1.71711	0.870166
1	1.33212	1.44622	0.180822	1.89394	2.008	0.754821
1	0.987302	1.74074	0.26045	2.30864	1.15508	0.837766
1	1.25954	1.23214	0.415152	1.21898	1.67665	0.75942
1	1.30672	1.49027	0.215434	2.9403	1.30457	0.62141
1	1.28195	1.46939	0.442815	0.735099	2.20721	0.738889
1	1.17949	1.58167	0.415761	1.27451	1.28718	0.785894
1	1.35135	1.45914	0.22	2.4026	1.38919	0.690667

```
|1  1.05822 1.31525 0.508091  1.13376  1.6573 0.752577|
└──────────────────────────────────────────────────────┘
```







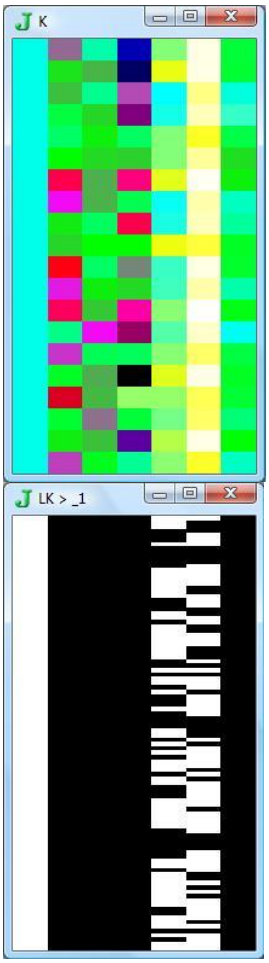


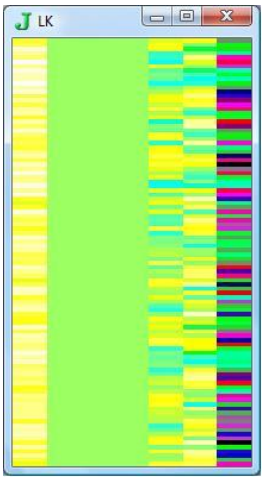
The third image focuses on the three columns of middle size numbers. The result is somewhat of a surprise as the numbers have all been chosen randomly. Actually this is possible. Can you understand how it could happen?

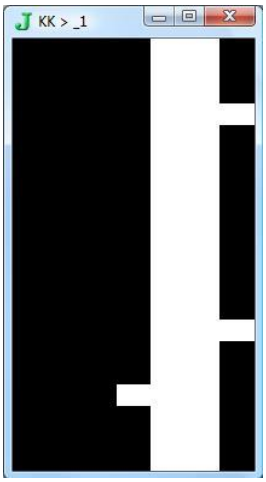
KK

```
┌──────────────────────────────────────────────────────┐
|_1 _1.59259 _1.09967  _2.66197 _0.761329 _0.221184  _1.27381|
|_1 _1.40217 _1.47674  _2.78788 _0.606299 _0.222222  _1.28571|
|_1 _1.46196 _1.14126  _1.64286 _0.970684 _0.368421  _1.02013|
|_1 _1.26316   _1.425  _2.34568 _0.935673 _0.286219 _0.884375|
|_1 _1.20725 _1.38627  _1.21384 _0.764706 _0.512903  _1.25506|
|_1 _1.36517 _1.41975  _1.43548 _0.762319 _0.333333  _1.41445|
|_1 _2.03053 _1.48496  _1.95522 _0.617722 _0.198225  _1.38525|
|_1 _1.74194 _1.48148     _1.24 _0.953125 _0.308642    _1.0623|
|_1 _1.39267 _1.21429  _2.03191 _0.928793 _0.279762      _1.12|
|_1 _1.42714 _1.36268  _1.36301 _0.599483 _0.481848  _1.30603|
|_1 _2.11811 _1.20818  _1.54878     _0.88 _0.221622  _1.29371|
|_1 _1.71711 _1.38697  _1.42056 _0.870166 _0.309249  _1.09841|
|_1    _2.008 _1.44622  _1.89394 _0.754821 _0.180822  _1.33212|
|_1 _1.15508 _1.74074  _2.30864 _0.837766  _0.26045 _0.987302|
|_1 _1.67665 _1.23214  _1.21898  _0.75942 _0.415152  _1.25954|
|_1 _1.30457 _1.49027   _2.9403  _0.62141 _0.215434  _1.30672|
|_1 _2.20721 _1.46939 _0.735099 _0.738889 _0.442815  _1.28195|
|_1 _1.28718 _1.58167  _1.27451 _0.785894 _0.415761  _1.17949|
|_1 _1.38919 _1.45914   _2.4026 _0.690667     _0.22  _1.35135|
```

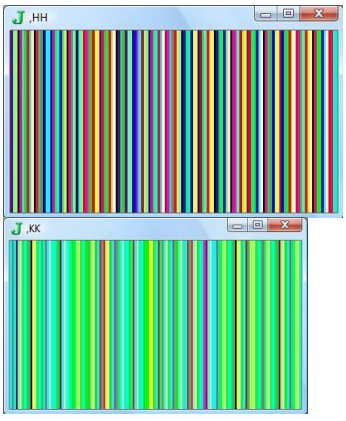
```
┌──────────────────────────────────────────────────────────────────────┐
│_1  _1.6573 _1.31525  _1.13376 _0.752577 _0.508091  _1.05822│
└──────────────────────────────────────────────────────────────────────┘
```

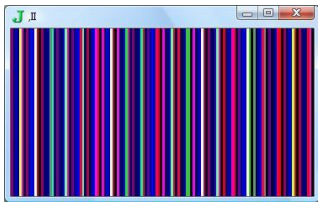


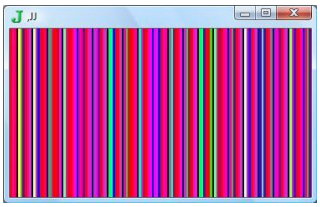




Here the third image shows where numbers are greater than _1 in KK. Then the fourth image show where numbers are greater than _1 in LK.

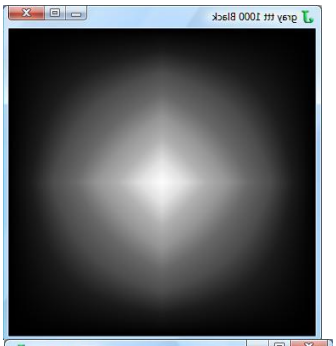


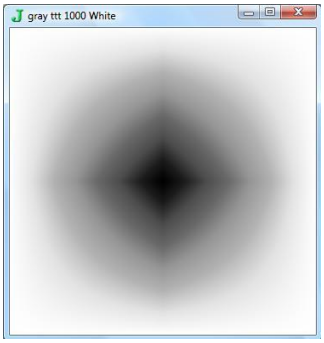


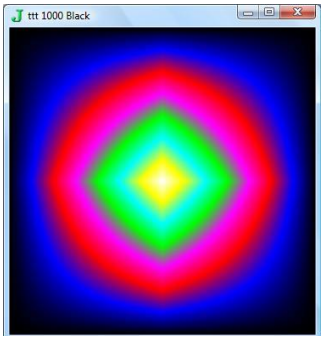


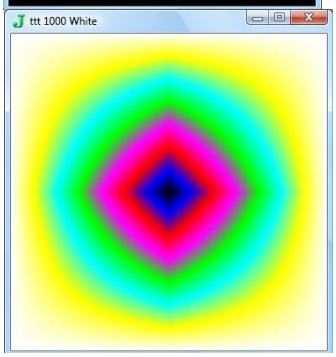
These four images above are a ravel of each of the four tables HH, II, JJ and KK. The first table is randomly distributed. Table II has a large proportion of small numbers so it appears quite blue. The one representing JJ has many middle values with a few small and large outliers and is mostly red. The values in KK are all relatively large and green is the dominant color.

ttt is a verb named “turn, turn, turn”. The first and third images have a black background. The second and fourth are white. All four are of a square inside a circle. If you look at the definition, you may understand how ttt got its name.









The joy of J is that a dialog is possible. This paper uses viewmat. It really needs only two nouns and four verbs. All the rest is basically a conversation which has endless possibilities.

```
load 'viewmat'
RGB=:( #:i.8){0 255
GRB=:1 0 2{"1 RGB
at2=: 13 :'([:{:"1 *.) j./"1 y'
gray=: 13 :'3#"0 +<.0.5+"0 +/"1]0.3 0.59 0.11*"1 y'
ttt=:(([: */~ i.) , [: |. [: */~ i.) ,. [: |: [: |. [: |: ([: */~ i.) , [: |. [: */~ i.
```

As promised, this is a capture of the dialog in the final section of the paper.

```
load 'viewmat'
RGB=:( #:i.8){0 255
PAL=: _1 }. 1 }. RGB
GRB=:1 0 2{"1 RGB
A=:10 +>:?20#100
B=:22+>:?20#100
C=:80+>:?20#100
D=:100+?20#100
E=:_100+?20#100
F=:_120+?20#100
G=:_140+?20#100
T=:200 + A,.B,.C,.D,.E,.F,.G
H=:(7?7){"1 T
I=:%"0(7?7){"1 H
J=:H*(7?7){"1 I
K=:-|(7?7){"1 J

ALL=:H;I;J;K

HH=:>0{ALL
II=:>1{ALL
JJ=:>2{ALL
KK=:>3{ALL
GRB viewmat HH;'H'
GRB viewmat II;'I'
GRB viewmat JJ;'J'
GRB viewmat KK;'K'
LA=:10 +>:?100#100
LB=:22+>:?100#100
LC=:80+>:?100#100
LD=:100+?100#100
LE=:_100+?100#100
LF=:_120+?100#100
LG=:_140+?100#100
LT=: 200 + LA,.LB,.LC,.LD,.LE,.LF,.LG
```

```
LH=:(7?7){"1 LT
LI=:%"0(7?7){"1 LH
LJ=:LH*(7?7){"1 LI
LK=:-|(7?7){"1 LJ
GRB viewmat LH;'LH'
GRB viewmat LI;'LI'
GRB viewmat LJ;'LJ'
GRB viewmat LK;'LK'

GRB viewmat (1 2 {"1 HH);'1 2 {"1 HH'
GRB viewmat (0 3 4 {"1 HH);'0 3 4 {"1 H H'
GRB viewmat (II<0.004);'II<0.004'
GRB viewmat((0.003<II)*.II<0.004);'0.003 < II < 0.004'
GRB viewmat (0 1 2 {"1 JJ);'0 1 2 { J J'
GRB viewmat(0 2 {"1 JJ);'0 2 { J J'
GRB viewmat ( _1<KK);'K K > _1'
GRB viewmat (,HH);',H H'
GRB viewmat (,II);',I I'
GRB viewmat (,JJ);',J J'
GRB viewmat (,KK);',K K'
ttt=:(([: */~ i.) , [: |. [: */~ i.) ,. [: |: [: |. [: |: ([: */~ i.) , [: |. [: */~ i.
gray=: 13 :'3#"0 +<.0.5+"0 +/"1]0.3 0.59 0.11*"1 y'
(gray GRB) viewmat (ttt 1000);'gray ttt 1000 Black'
(gray GRB) viewmat (999001-ttt 1000);'gray ttt 1000 White'
GRB viewmat (ttt 1000);'ttt 1000 Black'
GRB viewmat (999001-ttt 1000);'ttt 1000 White'
```