（ＪＡＰＬＡ 2008/08/2-4）

初めてさんのＪ言語（PART ）

統計数理研究所（名誉教授）　鈴木義一郎

《Ｊ言語の特徴について》－

ⅰ）Ｊ言語はAPLの“hybrid”であるが、機能がさらに拡大されている。特にAPLのように特殊文字を使うことなく、アスキー文字だけで利用できることが大きな特徴である。

ⅱ）Ｊ言語には「品詞」という概念があり、“何らかの演算を行う”という関数は、全て「動詞」である。

ⅲ）Ｊ言語には「アレイ」という概念が登場し、」いわゆる“ベクトル”や“行列”といった概念が拡張されている。

ⅳ）複数個の演算の「結合」に関しては、数学などで考えられているパターンを“拡張”した考え方で捉えられている。一般に、複数個の演算子の結合形に対しては「トレイン(train)」という言葉が使われている。

ⅴ）ほとんどの演算が、複素数に対しても適用可能である。

ⅵ）システムに組み込まれている「原始動詞(primitive verb)」が豊富である（中にはどんなときにどんな目的で利用すべきかが不可解なものもあるが……）。特に「逆行列演算[%.]」、「自己分類[=]の片側形」,「重複要素の排除[~.]」、「一般ガンマ関数[!]の片側形」、「円関数[o.]」、「書式関数[":]」などは、各種の数値計算に非常に強力である。さらに、「~」、「/」、「/.」、「\」、「^:」、「f.」といった副詞も、動詞を修飾していろいろな機能を持たせるときに極めて有用である。

ⅶ）強いて難点を挙げるならば、機能が高級すぎるが故に、その全容を理解することが非常に難解である。

《Ｊ言語で用いられる文字》

「e」という文字は、数値を指数表示するときに用いられる。

$2.4e3 \quad \Leftrightarrow \quad 2.4 \times 10^3 = 2400$

「r」という文字は、有理数の分数表示として用いられる。

$2r3 \quad \Leftrightarrow \quad 2/3 = 0.666667$

「b」という文字は、「基底」を表す機能を持っている。

$2b111 \quad \Leftrightarrow \quad 1\times 2^2 + 1\times 2^1 + 1\times 2^0 = 7 \quad ; \quad 16b1f \quad \Leftrightarrow \quad 1\times 16^1 + 15\times 16^0 = 31$

「p」という文字は、円周率「π」に関する数値を表す。

$1p1 \quad \Leftrightarrow \quad \pi = 3.14159 \quad ; \quad 2p1 \quad \Leftrightarrow \quad 2\pi = 6.28319$

「x」という文字は、オイラーの定数「e」に関する数値を表す。

$1x1 \Leftrightarrow e = 2.71828$ ； $2x1 \Leftrightarrow 2e = 5.43656$ ； $1x2 \Leftrightarrow e^2 = 7.38906$

「j」という文字は、複素数「$1+2i$」を「1j2」のように表す。

「ad」という文字は、左に絶対値を右に偏角を度数で与えて複素数を表示できる。

「ar」という文字は、左に絶対値を右に偏角をラジアンで与えて複素数を表示できる。

$5ad53.1301 \Leftrightarrow 3j4$ ； $5ar0.927295 \Leftrightarrow 3j4$

《アレイの概念と基本用語の解説》

J言語には、数学などで使われるベクトルや行列といった概念を拡張したものとして、「アレイ(array)」という概念が登場する。各アレイには、「ランク(rank)」と呼ばれているものが呼応している。

ランクが0のアレイをアトム(atom) …… スカラー(scalar)

ランクが1のアレイをリスト(list) …… ベクトル(vector)

ランクが2のアレイをテーブル(table) 行列(matrix)

アトム：「1.2」、「1j2」、「'a'」といった実数、複素数、文字などのユニットをアトムという

リスト：リストの形(shape)は、アトムの個数で与えられ、「$」という演算子で求めることができる。実際

```
   $ L1=:>:i.5

5
```

のように表示されるが、アトムに対しては

```
   $ 1.2

```

のように、空(` ')が与えられるだけである。つまりアトムは形が無いのである。

テーブル：L1というリストに「L2=:6 7 8 9 10」というもう1つのリストを0軸(縦)方向に連結するという接続詞「,:」で連結した

```
   ]T=:L1 ,: L2

1 2 3 4  5
6 7 8 9 10
```

は「テーブル」と呼ばれる。このTの形は

```
   $ T

2 5
```

のように表示され、0軸方向に2(行)、1軸方向に5(列)あることを示している。

ランク：形を求める「$」という動詞を 2 度続けてオペレートするという演算は「&」という接続詞で連結して、「rank=:$&$」のように定義される。この関数の実行例は

rank 1.2	rank L1	rank T	rank A=:i.2 3 4
0	1	2	3

のようになる。つまり、各アレイの形を求めてから、さらにその形（アトムの個数）を求めているもので、これが「ランク」に外ならない。

アイテムとタリー：アレイに対する「アイテム(item)」とは、アレイから 0 軸を除いた 1 軸以降のユニットをいう。従って、テーブルのアイテムはリストで、リストのアイテムはアトムである。ただアトムに対しては 0 軸を除いてしまうと何も無くなるので、アレイ自身がアイテムということになる。そして、「タリー(talley)」と呼ばれる演算子「#」がアイテムの個数を出力する。

# 1.2 1	「1.2」というアトムが 1 個
# L1 5	アトムというアイテムが 5 個ある。 リストに対してのみ「$」と同じ結果を与える。
# T 2	長さ 5 のリストが 2 本ある。
# i.2 3 4 2	3×4 のテーブルが 2 枚ある。

《J 言語の「品詞」について》

J 言語における関数の定義式は

*'area = .3.14 * * :radius'*

のように表現されるが、対応する英語表現は

「 *area is* 3.14 *times square of radius.* 」

まず「名詞」とは、ユーザーが定義するデータ（文字型変数やボックス型変数も含めて）がそうであり、演算結果なども全て名詞である。さらに、

```
   ]W=:(<'I'),(<'am'),(<'a'),<'Japanese.'
```

I	am	a	Japanese.

のようなボックス表現も名詞の部類に入る。ここで「<」は「ボックスで囲む」という動詞で、「]」は「右側の内容を表示する」という動詞である。

次に「動詞」は、名詞に作用して結果も名詞である。なお、動詞の作用を浮ける名詞は「引

数」と呼ばれ、さらにほとんどの動詞は

右引数にだけ作用する場合：片側形(monadic)

両側の引数に作用する場合：両側形(dyadic)

のように２通りの演算機能がある

Ｊ言語に組み込まれている動詞は、極めて豊富である。またこのような組み込み関数を使って「sum=:+/」のように自分流に定義しておけば、意味を解釈しながら縦横に利用することができる。このユーザーが定義した「sum」のような関数を「代動詞(proverb)」と名づけている。組み込み関数にしろユーザーが定義したものにしろ、多くのものは、ある行為を行って何らかの結果を得るという意味で、ほとんどのものが「動詞」である。つまり、コンピュータを使うこと自体が“動詞”なのである。

次に「副詞」は、名詞や動詞を右側から修飾し、結果は名詞や動詞となる。たとえば、「*」という動詞に「/」という副詞で修飾して「*/」という動詞が得られる。これを片側形として用いると、引数の全ての要素の積を与える関数となる。また両側形として用いると、次のような乗算のクロス表(外積)が出力される。

```
   I */ I=:>:i.9

1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

これは、掛け算の「九・九の表」に他ならない。

さらに「接続詞」は、必ず両側形として用いられ、２つの名詞を連結して名詞を作ったり動詞どうしを連結して新たな動詞を生成したり、さらに動詞と名詞を連結して動詞を作るなど、多様な働きをする。

たとえば、「%」「!」という２つの動詞を「&」という接続詞で連結した「%&!」という関数は、片側形で用いると

```
   %&! 3

0.166667
```

といった結果をが得られる。これは「!3=6」の逆数を与えている。また両側形では

```
   5 %&! 3
20
```

という結果で、これは「(!5)%!3=20」で、2項係数 $_5C_3 = \frac{!5}{!3} = \frac{120}{6} = 20$ に他ならない。

さらに、動詞と名詞との連結の例では

2&*	*&2 L1	+: L1
L1=:>:i.5	2 4 6 8	2 4 6 8
2 4 6 8 10	10	10

といったように、順番はとちらでも同じで、これは「2倍にする」という原子動詞「+:」と同じ演算結果を与える。

《原始定義動詞の特徴》

ⅰ）J言語では、論理演算に関しては非常に“寛容”である。たとえば

```
   *: %: 2
2
```

のように、「%:」で２の平方根をとってから「*:」で平方した値はピタリ２になる。つまり、コンピュータによる丸めの誤差を吸収して、論理的整合性を保つように設計されている。さらに「0 の 0 乗」も論理的に「1」が対応するようになっている。

ⅱ）｛+.：*.：-.｝という演算子が、数値に対しては｛最大公約数：最小公倍数：補数を与えるが、同じ演算子が論理演算ではそれぞれ｛論理和：論理積：論理否定｝に対応していて、記号の整合性が保たれている。

ⅲ）J言語の演算はすべて、複素数にまで拡張されている。たとえば「*」の片側形は複素平面上の単位円周上の点に射影する演算子であるが、これは実数に対して符号を与えるという機能の“拡張形”になっている。ただ順序に関連した演算子を複素数に適用すると、’domain error’となることが多い。

ⅳ）「!」という演算子は、自然数だけでなく有理数や負の数に対しても適用できて、いわゆる「一般ガンマ関数」を与えるものである。

ⅴ）三角関数、双曲線関数、円関数、楕円関数などはすべて「サークル関数」と呼ばれていて、「o.」という演算子の両側形として与えられる。左引数に奇(偶)数を入力すれば奇(偶)関数、符号を変えれば逆関数を与えるといった整合性も保たれている。たとえば「1&o.」は「sin 関数」で、「2&o.」は「cos 関数」である。

ⅵ）正方行列の逆行列を与える「%.」を両側形として使えば、線形関係式の最小２乗解が出力される。

《J言語における合成関数》

J言語で用いられる関数は、片側形だけとは限らず両側形も扱われる。つまり、2つの関数 g,f が共に片側形の場合には、「$(g * f)y = g(f\ y)$」という結合だけで済まされるが、関数 g が両側形の場合も想定してみると、「$(g * f)y = y\ g(f\ y)$」といった結合の仕方も考えれる。これが、片側形の場合の「フック(hook)」と呼ばれている概念に他ならない。さらに、合成関数 $g * f$ が両側形として機能する場合には、

① $g(x\ f\ y)$ ……… f が両側形で g が片側形の場合

② $(f\ x)\ g(f\ y)$ ……… f が片側形で g が両側形の場合

③ $x\ g(f\ y)$

のように、３通りの結合の仕方が考えられる。そこでJ言語では

① $x(g @ f)y = g(x\ f\ y)$

② $x(g \& f)y = (f\ x)\ g(f\ y)$

③ $x(g\,f)y \;=\; x\;g(f\;y)$

のような２つの接続詞を用いて区別することにしたのである。この③の結合の仕方が両側形の場合の「フック(hook)」となる。なお片側形の場合も

$(g@f)y = (g\,\&\,f)y = g(f\;y)$

$(g\;f)y \;=\; y\;g(f\;y)$

のように表記して、フックとそうでない場合とを区別する。

さらに、2 つの片側関数｛g,h｝と両側関数g が与えられているときに、これら３つの動詞の結合ルールとして

$(f\;g\;h)y = (f\;y)g(h\;y)$ ..... 片側形の場合

$x\;(f\;g\;h)\;y = (f\;x)g(h\;y)$ ..... 両側形の場合

という合成関数が定義される。これが「フォーク(Fork)」と名づけられているものである。「mean=:+/%#」という関数がフォークの典型例である。

《演算の結合ルール》

ｉ）演算の順序は、原則として右から左に実行する。加減より乗除が先といったような優先順位はない。

ii）接続詞、次いで副詞は、動詞に先がけて連結される。たとえば、

```
   *&+/ L1=:>:i.5      *&(+/) L1

120                    1
```

といった演算結果からも分かるように、「*&+/」は「(*&+)/」であって「*&(+/)」とは働かない。

iii）動詞は、それが可能である限り，両側形として機能する。たとえば、

```
   +/2*L1=:>:i.5       +&2* L1

30                     3 3 3 3 3
```

といった演算結果からも分かるように、「+/2*L1」では、「/」という副詞は左にある「+」という動詞と先に連結し、その結果「2* L1=2 4 6 8 10」のように「*」も両側形として機能している。

また「+&2* L1」では、「+&2」の部分先に連結するから「*」は片側形として機能して「*L1=1 1 1 1 1」のように演算する。

iv）とにかく、名詞や動詞に副詞や接続詞がついて演算の順序が影響を受けると懸念される場合には、先に優先させて演算したい部分をカッコで囲むことが必要になる。

局所定義は　イコールピリ(=.)　イコールコロン(=:)で　大局定義

【局所定義と大局定義】

```
   test=:3
:0

a=.i.y
a;b=:1+a
)
```

```
    test 5
┌─────┬─────┐
│0 1 2 3 │1 2 3 4 │
│4       │5       │
└─────┴─────┘
(i.5);1+i.5
┌─────┬─────┐
│0 1 2 3 │1 2 3 4 │
│4       │5       │
└─────┴─────┘
```

左側のボックスで「test」という片側形の関数を定義して、引数に「5」を挿入して実行した結果が右側のボックスで示される。

「;(セミコロン)」は左右の引数をボックスで囲んで接続する動詞である。

```
   a
|value  error:
a
```

```
   b
1 2 3 4 5
```

関数の実行後、イコールピリで定義した「a」は消え、イコールコロン(=:)で定義した「b」は残っている。

定義関数内で大局定義を用いると、関数の実行後にいろんな変数が残ってしまって煩雑になる恐れがある場合には、局所定義を用いたほうがベターである。

【タシット(Tacit)定義とエクスプリシト(Explicit)定義】

タシット(Tacit)で　定義するのが　醍醐味さ　J特有の　面白さ

演算を　右から順に　作動さす　ことも可能さ　エクスプリシト(Explicit)

《簡単で、しかし重宝ないくつかの関数を定義してみよう！》

```
T=:3 5 3 2 4 6 4 5
```

8個のテストデータを「T」に入力する。

```
mean=:3 :'(+/y)%#y'

mean_t=:+/%#
```

```
   mean T
4
```

{mean」がExplicit、「mean_t」はTacitによる関数の定義。

```
dev=:3 :'y-mean y'

dev_t=:-mean_t
```

```
   ]d=:dev T
_1 1 _1 _2 0 2 0
1
```

「(平均からの)偏差：d」を出力する関数

```
var=:3 :'mean*:dev y'

var_t=:[:mean[:*:dev_t
```

```
   var T
1.5
```

偏差:d の平方値の平均値、つまり「分散」を求める関数である。

```
sdev=:3 :'%var y'

sdev_t=:[:%:var_t
```

```
   sdev T
1.22474
```

分散の平方根、つまり「標準偏差」を求める関数である。

「+/y」は合計値、「#y」は個数、「x%y」は割算、「*:」は平方値、「%:」は平方根

動詞が３つ　並んだときは　左右が先で　中の動詞は　３番手(フォーク Fork)

【タシット(Tacit)で関数を定義するには、フォーク(Fork)の概念をマスターすべし！】

]S=:+/T	]N=:# T	S % N	(+/%#)T	中の両側動詞を挟んで３つの連結動詞「fgh」をフォーク(Fork)。
32	8	4	4	

形なき　たったひとつは　「アトム」なり　アトムが並んで　「リスト」を作る
テーブルの　「形」を示す　ドル($)マーク　アイテム数は　シャープ(# talley)さん
引数の　低次のランクの　全てのものを　「１セル」「２セル」　などと呼ぶ
演算は　１つ低次の　セル相手　これを名づけて　「アイテム」と呼ぶ

【Ｊ言語と数学用語との対応表】

Ｊ言語	数学用語	Ｊでの表示例	ランク
アトム	スカラー	2	0
リスト	ベクトル	2 4 6(2*1+i.3)	1
テーブル	マトリクス	1 2 3 4 5 6(1+i.2 3)	2
レポート	多次元配列	i.2 2 3 0 1 2 3 4 5 6 7 8 9 10 11	3

【「$」Shape(アレイの形)：「#」Talley(アイテム数)】

]A0=:2 2]A1=:>:i.5 1 2 3 4 5	]A2=:>:i.3 4 1 2 3 4 5 6 7 8 9 10 11 12	shape=:$@] item=:#@] rank=:$&$]A3=:>:i.2 3 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24	「行列論」でスカラーに相当するA0が「アトム」、横ベクトルが「リスト」A1で、いわゆるA2のような行列が「テーブル」である。 A3のように、ランクが３（以上）のものは、「一般アレイ」と呼ばれる。
(shape;rank)A0 0（アトムの形は「空」）	(shape;rank)A1 5 1 （アトムが５個）		
(shape;rank)A2 3 4 2 （形４のリストが３本）	(shape;rank)A3 2 3 3 4 （３× ４のテーブルが２枚）		
item A0 1	item A1 5	item A2 3	item A3 2

【「item e」はアイテム（引数の１つ低次のランクのセル）の要素を出力する関数】

item_e=:<"_1 item_e A0 2 ‘ ‘ item_e A1 1 2 3 4 5	,. item_e A2 1 2 3 4 5 6 7 8 9 10 11 12	,. item_e A3 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24	引数の低次のランクの全てを「１セル」「２セル」などと呼ぶ。 演算は１つ低次のセルが相手で、これを名づけて「アイテム」と呼ぶ。

【いろいろな次数の「セル」を出力してみせている】

cell0=:<"0@]

cell1=:<"1@]

cell2=:<"2@]

cell3=:<"3@]

cell0 A0

2 cell0 A1

1	2	3	4	5

(0 次のセルを出力)

cell0 A2

1	2	3	4
5	6	7	8
9	10	11	12

(0 次のセル)

cell0 A3

1	2	3	4
5	6	7	8
9	10	11	12

13	14	15	16
17	18	19	20
21	22	23	24

(A3 の 0 次のセルは 2×3×4＝24 個ある)

cell2 A2

1	2	3	4
5	6	7	8
9	10	11	12

(２次のセルは引数)

cell2 A3

2	2	3	4	13	14	15
6	6	7	8	16		
9	10	11	12	17	18	19
				20		
				21	22	23
				24		

(２次のセルは「アイテム」と一致)

cell1 A0

2 cell1 A2

1	2	3	4	5	6	7	8	9	10	11	12

(ランク１のセルを出力している)

cell1 A3

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24

(2×3×4 というアレイの１次のセルは６個)

【ランクを指定しない場合やランクを指定した場合の演算結果】

+/ A0	+/ A2	+/ A3	+/"1 A3
2	15 18 21 24	14 16 18 20 22 24 26 28 30 32 34 36 (面と面の間に「+」が挿入される。)	10 26 42 58 74 90
+/ A1	+/"1 A2		+/"2 A3
15	10 26 42		15 18 21 24 51 54 57 60
+/"1 A1	+/"2 A2		(縦方向の和)
15	15 18 21 24		
ランク指定無と同じ	ランク指定無と同じ		

「]」は右で定義した変数を表示する。「”1」は次数(rank)１の引数に作用させる「副詞」

「# y」はアイテム(引数ｙより１つランクの低いセル)の数を出力する。

「J 言語」は“高級電卓”である！

【「+(Plus)」：左引数の値と右引数の値を足算する】

5 + 1 2 3 6 7 8	1 2 3 + 4 5 6 5 7 9	3r5 + 1r2 11r10	「arb」は「a/b」という分数で、これを利用すれば分数計算で悩むことはない。
3j4 + 1j1 4j5	3j4 + 1j_1 4j3	3j4 + 3j_4 6	

【「-(Minus)」：左引数の値から右引数の値を引く】

5 - 1 2 3 4 3 2	1 2 3 - 4 5 6 3 3 3	3r5 - 1r2 1r10	足算・引算が複素数でも大丈夫だから、電卓より高級ジャン
3j4 - 1j1 2j3	3j4 - 1j_1 2j5	3j4 - 0j4 3	
1 2 3 -~ 4 5 6 3 3 3	1j1 -~ 3j4 2j3	「x-~y」は「y-x」と同じ演算結果を出力 「~」は左右の引数を反転させる「副詞」	

【「*(Times)」：左引数の値と右引数の値を掛算する】

2 * 1 2 3 2 4 6	1 2 * 4 5 4 10	3r5 * 1r2 3r10	分数同士の掛算となると、存外、厄介なもの。 （J言語バンザ～イ）
3j4 * 1j1 1j7	3j4 * 1j_1 7j1	1j1 * 1j_1 2	
3j4 * 0j1 4j3	3j4 * 0j_1 4j_3	「0j1」を掛けると虚部の符号を変えて実部へ 「0j_1」を掛けると実部の符号を変え虚部へ	

【「%(Devide)」：左引数の値を右引数の値で割る】

2 4 6 % 2 1 2 3	2 8% 4 0.4 0.5 20	2 % 1j1 1j 1	２や５は、実数の世界では分解できない「素数」だが、複素数ならば分解デキルノダ！
]z=:3j4%1j1 3.5j0.5	z * 1j1 3j4	5 % 2j1 2j 1	
2 %~ 2 4 6 1 2 3	1j1 %~ 3j4 3.5j0.5	「x%~y」は「y%x」と同じ演算結果を出力 「~」は左右の引数を逆転させる「副詞」	
3j4 * 0j1 4j3	3j4 % 0j_1 4j3	「z* 0j1」と「z% 0j_1」は同じ結果になる！ 「0j1」と「0j_1」の偏角は符号が反対だから、「割算」の場合マイナスの偏角を引くことになり、結果としてプラスになる。	
*.0j1 1 1.5708	*.0j_1 1 1.5708		

【J言語には、「名詞(0)」、「副詞(1)」、「接続詞(2)」、「動詞(3)」といった品詞がある】

動詞との　出会いひたすら　待つ「副詞」　右にこだわる　「接続詞」

オープン(>)は 動詞でアンド(&)は 接続詞　イーチ(&> &.>)にすれば　副詞に変身

【定義内容の品詞は「4!:0<'def'」で識別：「0：名詞、1：副詞、2：接続詞、3:動詞」】

a=:10 4!:0<'a' 0	av=:/ 4!:0<'av' 1	c=:& 4!:0<'c' 2	mean=:+/%# 4!:0<'mean' 3

mean=:+/%# sum=+/ （２つの動詞を定義）	4!:1(3) mean sum	namelist 3 mean sum	4!:55<'mean' 1 >4!:1(3) sum	erase'sum' 1 $namelist 3 0

「4!:1(3)」は「namelist y」と同じ。「4!:55<'y'」は「erase'y'」と同じである。

]d=:1 2 3;4 5 1 2 3 4 5	+/L:0 d 6 9	sum&.> d 6 9	+/&> d 6 9	sum=:+/ 4!:0<'sum' 3
mean L:0 d 2 4.5	mean&.> d 2 4.5	mean&> d 2 4.5	mean each d 2 4.5	each=:&> 4!:0<'each' 1

「;(セミコロン)」は左右の引数をボックスで囲み接続する両側動詞(Link)である。

【単項(monadic)と二項(dyadic)】

ほとんどの動詞には、(右引数だけの)単項(monadic)と、(左右に引数をとる)二項(dyadic)の２種類がある。

]nl=:^.100 4.60517	^ nl 100	]ol=:10 ^. 100 2	3 :'1x1 ^.y'100 4.60517

「^.」の片側形は(自然)対数関数で、「^」の片側形は指数関数(^.の逆関数)

1x1 2.71828 o.1 3.14159	]b=:1x1 ^. 100 4.60517 10 ^. 100 2	1x1 ^ b 100 10 ^ 2 100	「^.」の両側形は左引数を底とする対数関数「1x1」はオイラーの定数 「1x1^.y」は「^.y」と同じ結果を出力する。「o.1」は円周率(π)

x:o.1 1285290289249r409120605684	x:1x1 6157974361033r2265392166685

[x:] は 倍精度の演算

20r30	2r3 + 1r5	2r3 - 1r5	[r]は「分数」を与える特殊な名詞で、左右の数を離してはいけない。

2r3	13r15	7r15	これを使えば分数計算もラクチン

【名詞】

名詞の種類としては、①数値、②文字、③ボックス(Box)の３種類である。

```
odd=:1 3 5

even=:2 4 6
```

数値はそのまま定義する。

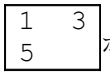
```
   even - odd
1 1 1
```

```
SEI=:'SUZUKI'

Mei=:'Giitiro'
```

クオート(` `)で囲み定義

```
   SEI,' ',Mei
SUZUKI Giitiro
```

```
   ]b=:<odd
```



ボックスと数値は直接演算はできないが、ボックス内では(L:0)を使い演算は可能

【Ｊ言語のプリミティブ(+,*:等)】

Ｊ言語では、「+」や「*:」等は「プリミティブ(Primitive)」と呼ばれ、動詞、副詞、接続詞等がある。演算用のアスキー記号をそのまま使ったもの(ベア)、ピリオド「.」をつけたもの、さらにコロン「:」を付したものと３種類あり、機能はそれぞれ異なる。

プリミティブの用法は固定されているが、「plus=:+」や「squre=:*:」等のように、名前をつけて使うこともできる。このときの「plus」や「squre」は「代動詞」と呼ばれる。

```
   3 + 5
8
   3 plus 5
8
```

```
   *: 1 2 3
1 4 9
   squre 1 2 3
1 4 9
```

プリミティブと数値の間には「スペース」をとらなくともよいが、代動詞と数値の間には必ずスペースをとる必要がある。

```
   + 3.14
3.14
   + 3j4
3j_4
```

```
   0 0 1 1 *: 0 1 0 1
1 1 1 0
   0 0 1 1 +: 0 1 0 1
1 0 0 0
```

「*:」の両側形は「否定論理積」である。
(左右の引数が共に１のときだけ０を出力)
「+:」の両側形は「否定論理和」である。
(左右の引数が共に０のときだけ１を出力)

計算を　マトメテ演算　したければ　レベル(L:0)やイーチ(&>)を　使えばよい

【レベル(L:0)という副詞を用いた計算例】

```
X2=:1+X1=:1  2
3

Y2=:1+Y1=:2  1
3

regb=:[%.1:,.]
```

```
    Y1 regb
X1

1 0.5
```

```
    Y2 regb
X1

2 0.5
```

```
    Y1 regb
X2

0.5 0.5
```

```
   Y2 regb X

2 1.5 0.5
```

上のボックス内に、4組のデータに対する回帰直線が出力されているが、「L:0」という副詞を用いると、以下のように4組のデータの計算結果を同時に出力することができる。

```
            (Y1;Y2;Y1;Y2)regb     L:0
X1;X1;X2;X2
┌──────┬──────┬──────┬──────┐
│1 0.5 │2 0.5 │0.5 0.5│1.5 0.5│
└──────┴──────┴──────┴──────┘
(Y1;Y2;Y1;Y2)regb&.> X1;X1;X2;X2
┌──────┬──────┬──────┬──────┐
│1 0.5 │2 0.5 │0.5 0.5│1.5 0.5│
└──────┴──────┴──────┴──────┘
```

```
(Y1;Y2;Y1;Y2)regb&> X1;X1;X2;X2
1 0.5
2 0.5
0.5 0.5
1.5 0.5
```

「L:0」の代わりに「each =:&>」という副詞を用いると、ボックスを外した結果を出力。

【フォーク(Fork)とフック(Hook)は単項と二項のケースがあり、働き方は複雑だ！】

動詞が３つ　並んだときは　左右が先よ　中の動詞は　３番手(フォーク Fork)
片側動詞に　両側動詞が　連結すれば　カッコでくくり　これ「フック(Hook)」
並んだ動詞は　右からフォーク　残った動詞で　またフォーク(フック)

【J言語の特徴：Fork と Hook】

```
   (sum=:+/)D=:3 1 2
6
```

```
   ]S=:+/D
6
```

```
   ]N=:# D
3
```

```
   S % N
2
```

```
   (mean=:+/%#)D
2
```

中の両側動詞を挟んで３つの連結動詞「fgh」をフォーク(Fork)という。２連動詞「gh」はフック(Hook)

《フォーク(fgh)》

【単項】

```
   g
  ↙ ↘
 f   h
 ↓   ↓
 y   y
```

【二項】

```
     g
   ↙   ↘
  f     h
 ↙ ↘   ↙ ↘
 ↓   ↓ ↓   ↓
 X   y x   y
```

《フック(gh)》

【単項】

```
   g
  ↙ ↘
 y   h
     ↓
     y
```

【二項】

```
   g
  ↙ ↘
 x   h
     ↓
     y
```

```
  (]-+/%#)D=:1+i.5
2 1 0 1 2
```

```
   (]-mean)D
2 1 0 1 2
```

```
  ([:*:]-mean)D
4 1 0 1 4
```

「[:(cap)」は「何もしない」という動詞

```
   ]d=:(-+/%#)D
2 1 0 1 2
```

```
  (dev=:-mean)D
2 1 0 1 2
```

```
   ([:mean*:)d
2
```

```
   ([:mean|)d
1.2
```

「５連動詞」は、右端の３つの動詞でフォークを作って１つの動詞となり、これと左端の２つの動詞と連結してフォークとなる。また「４連動詞」は、右端の３つの動詞でフォークを作って１つの動詞となり、これと左端の１つの動詞と連結してフックとなる。

```
  ([:mean[:*:]-mean)D
2
```

```
  (var=:[:mean[:*:dev)D
2
```

```
   (sdev=:[:%:var)D
1.41421
```

```
      3  :'mean|y-mean
y'D
```

```
(mdev=:[:mean[:|dev)D
1.2
```

「sdev」は標準偏差

「mdev」は平均偏差

1.2		
「|」の片側形は、右引数の数値の絶対値を出力する関数である。		

【「エクスプリシト」から「タシット」へ】

13 :'(+/y.)%#y.' +/ % #	13 :'mean *:dev y.' [: mean [: *: dev	13 :'mean|dev y.' [: mean [: | dev
エクスプリシトで記述したプログラム(program)を、「13 :'program'」によってタシットに変換できる。		

【「+」:「+.」:「+:」】

実数に　プラス(+)の片側　そのままで　両側形は　フツーの足算

プラス・ピリ(+.)の　両側形は　最大公約数(GCD)　片側形は　実部と虚部

プラスコロン(+:)　右引数を　倍にする　両側形は　論理演算(否定論理和)

【「+」Conjugate(共役複素数)・Plus(足算)】

+ 0.4 0 3 0.4 0 3	+ 3j4 3j 4	複素数に対しては「+」の片側形は共役複素数を出力。実数にはソノマンマ
1 2 3 + 4 5 6 5 7 9	2j3 + 1 1j1 3j3 3j4	「ajb」と「cjd」の和は「(a+c)j(b+d)」 (実数部同士、虚数部同士で足算を行う)

【「+.」Real-Imaginaly(実部と虚部)・GCD(最大公約数)】

+. 5 5 0	+. 3j4 3 4	「+.」の片側形は実数部と虚数部を出力	
6 +. 8 2	6 +. 5 1	「+.」の両側形は最大公約数(GCD)を出力	
]t=:1+i.a=:6 1 2 3 4 5 6	t | a 0 0 0 2 1 0	「|」の両側形は剰余を出力する。	
t res a 0 0 0 2 1 0	t RES a 0 0 0 2 1 0	res=:4 :'x*t-<.t=.y%x' RES=:[*[(-<.)@]%~ NB,[-<.]:Hook	
]s=:0=t|a 1 1 1 0 0 1	s # t 1 2 3 6	div=:3 :'(0=t|y)#t=:1+i.y' (全ての約数を出力する関数)	
]c=:div a=:6 1 2 3 6	]d=:div b=:8	]t=:(c e.d)#c 1 2	{:/:~t 2

	1 2 4 8		
6 gcd 8 2	6 gcd 5 1	gcd=:4 :'{:/:~(a e. div y)#a=.div x' GCD=:div@[([:{:@/:~e.#[)div@] (「最大公約数(GCD)」を出力する関数)	

【「+:」Double(倍増)・Not-And(否定論理和)】

]d=:+: c=:1 2 3 2 4 6	2 * c 2 4 6	2&* c 2 4 6	*&2 c 2 4 6
-: d 1 2 3	-:b._1 +:	「-:」は「+:」の逆演算である。「b.」は副詞で「vb._1」でvの逆演算を出力する。	
(a=:0 0 1 1)+:b=:0 1 0 1 1 0 0 0	a(0:=+)b 1 0 0 0	「+:」の両側形は「否定論理和」で、左右の引数が共に「0」のときだけ「1」を出力	
a *: b 1 1 1 0	a(0:=*)b 1 1 1 0	「*:」の両側形は「否定論理積」で、左右の引数が共に「1」のときだけ「0」を出力	

【「-」:「-.」:「-:」】

マイナス(-)の 片側形は 符号の反転　両側フツーの　引算よ

マイナスピリ(-.)の　片側形は　足して1になる　「補数」を出力
マイナスピリ(-.)の　両側形は　「差集合」を出力す
マイナスコロン(-:)の　片側形は　右引数を　半分にする(倍増演算子「+:」の逆演算)
マイナスコロン(-:)の　両側形は　ソックリ同じなら　1を出力(論理演算)

【「-」:Negative(逆符号)・Minus(引算)】

- _5 0 2 5 0 2	- 3j4 3j_4 3j 4 3j4	+ _5 0 2 5 0 2	+ 3j4 3j_4 3j 4 3j4
4 5 6 - 1 3 2 3 2 4	3j4 - 1j1 2j3	マイナス「-」の 片側形は符号を逆にする。 プラス「-」の 片側形は共役複素数	

【「-.」:Not(補数)・Less(差集合)】

]b=:-.a=:4 0.3 1	a + b	]d=:-. c=:3j4	c + d

3 0.7 0	1 1 1	2j 4	1
(1:-]) a 3 0.7 0	(1:-]) c 2j 4	「-.」の片側形は、足して１になる「補数」を出力する。	
div=:3 :'(0=t|y)#t=:1+i.y'：全ての約数を出力する。			
]a=.div 6 1 2 3 6	]b=.div 8 1 2 4 8	]c=:a -. b 3 6	]d=:b -. a 4 8
]e=:(a e. b)#a 1 2	e, c 1 2 3 6	e, d 1 2 4 8	a -: e, c 1
「x -. y」はｙに含まれないｘの要素(差集合)を出力する。			

【「-:」Halve(半減)・Match(一致：論理演算)】

]b=: -:a=:2 4 6 1 2 3	a % 2 1 2 3	%&2 a 1 2 3	*&0.5 a 1 2 3
(halve=:%2:)a 1 2 3	]d=:-:c=:3j4 1.5j2	+: b 2 4 6	+: d 3j4
-:b._1 +:	+:b._1 -:	「-:」は「+:」の逆演算である。	
1 1 1 -:1 2 1 0	1 1 1 = 1 2 1 1 0 1	[-: match]の両側形はソックリ同じときだけ１を出力する。[=]の両側形は個別に等しいところに「1」を与える。 「+:」の両側形は、左右の引数が共に「0」のときだけ「1」を出力し、そうでなければ「0」を出力(否定論理和)。	
1 2 1 -: 1 2 1 1	1 2 1 = 1 2 1 1 1 1		
0 0 1 1 +: 0 1 0 1 1 0 0 0	1 2 1 +: 1 2 1 domain error		

【「*」:「*.」:「*:」】

シグナム(*)は　符号与える　片側形　複素数には　単位円に射影

スター･ピリ(*.)　複素数には　大きさと　偏角与える　片側関数

スター･ピリ(*.)の　両側形は　最小公倍数　複素数でも　イッツオーケー

スターコロン(*:)の　片側形は　倍増演算子　マイナスコロン(-:)の　逆演算

スターコロン(*:)の　両側形は　左右が１なら　０を出力(否定論理積)

【「*」:Signum(単位円)・Times(掛算)】

`   * _3 0 2` `1 0 1`	`   (|,])* 3j4` `1 0.6j0.8`	「3j4」という複素数を複素平面の単に円上に射影すると「0.6j0.8」で、絶対値は１。	
`   1 2 3 * 2` `2 4 6`	`   1 2 * 4 5` `4 10`	`   1 2 * 3 4 5` `length error`	左右の引数の「積」を出力する。

`      3j4*  0j1`
`0j_1`
`4j3 4j 3`
	`      3j4*  1j1`
	`1j_1`
	`1j7 7j1`
		`      3j4  %  0j1`
		`0j_1`
		`4j 3  4j3`
			「z*0j1」と「z%0j_1」の結果は同一である。

【「*.」: Length-Angle(絶対値と偏角)・LCM(最小公倍数)】

`   *. 2` `2 0`	`   *. _3` `3 3.14159`	実数の「正数」には偏角は０で、「負数」の偏角は「π＝3.14159」である。 一般の複素数に対しては、絶対値と偏角の値を出力する。			
`   *. 3j4` `5 0.927295`	`   *._3j0` `3 3.14159`				

`  ]c=:(|a=:4)<.|b=.6`
`4`
	`   ]m=:1+i.>.c`
	`1 2 3 4`
		`   ]s=:a*m`
		`4 8 12 16`
			`   ]t=:b*m`
			`6  12  18`
			`24`
				`(t e.s)#t`
				`12`	

`lcm=:4 :'(*x*y)*(b e.s*m)#b=.t*m=.1+i.>.(s=.|x)<.t=.|y'NB. 最小公倍数`					
`   4 *. 6` `12`	`   4 lcm 6` `12`	`   4 *. _6` `12`	`   4 lcm _6` `12`		
`  _4 *. _6` `12`	`   _4 lcm _6` `12`	左右の引数の最小公倍数(LCM)を出力			
`   1j1 *.3j1` `3j1`	`      1j1  *  3j1` `2j4`	`   /:~3j1 2j4` `2j4 3j1`	「*.」は複素数にも公倍数を出力するが、最小かどうかは疑問		
`   0 0 1 1 *.0 1 0 1`	`   0 0 1 1(*@*)0 1 0 1`	左右の引数が共に「1」のときだけ「1」を出力する(論理積)。			

0 0 0 1	0 0 0 1	

【「*:」：Square(平方値)・Not-And(否定論理積)】

]b=:*:a=:3 4 9 16	%: b 3 4	2 %: b 3 4	]d=:*:c=:1j1 2j1 0j2 3j4	%: d 1j1 2j1	2 %: d 1j1 2j1
0 0 1 1 *:0 1 0 1 1 1 1 0		0 0 1 1(0:=*)0 1 0 1 1 1 1 0		左右の引数が共に「1」のときだけ「0」を出力する(否定論理積)。	

【「%」:「%.」:「%:」】

パーセント(%)　片側形は　逆数で　両側形なら　フツーの割算

行列の　割算行う　パーセントピリ(%.)　片側形なら　逆行列

パーセントコロン(%:)　片側形なら　平方根　両側形は　累乗根

【「%」:Reciprocal(逆数)・Devide(割算)】

]b=:% a=: 2 _0.5 0.5 2	1 % 2 _0.5 0.5 _2	b * a 1 1	「%」の片側形は逆数を出力する。
]d=:% c=:1j1 3j4 0.5j 0.5 0.12j 0.16	1 % c 0.5j 0.5 0.12j 0.16	d * c 1 1	
*. 3j4 5 0.927295	*.% 3j4 0.2 0.927295	極座標表示では、逆数の絶対値は元の複素数の絶対値の逆数、偏角は符号が反対	

【「%.」: Inverse (逆行列)・Devide for Matrix(行列の割算)】

]A=:2 2 $ 1 1 0 1 1 1 0 1	]B=:%.A 1 _1 0 1	A +/ .* B 1 0 0 1	%. B 1 1 0 1
「鶴と亀の頭が 14 個で、足が 40 本である。鶴と亀はそれぞれ何匹づついるか？			
]C=:2 2 $ 1 1 2 4 1 1 2 4 (「鶴亀算」の行列)	14 40 %. C 8 6 (行列算で答は簡単)	]D=:%. C 2 _0.5 _1 0.5 (D は C の逆行列)	%. D 1 1 2 4 (D の逆行列は元の行列 C に戻る)
C +/ .* D 1 0 0 1	C +/ .* 8 6 14 40	「+/ .*」は行列と一般アレイ(リストやテーブル等)の掛算を行う。	
]I=:=/~k=:i.2 1 0 0 1	k =/ k 1 0 0 1	+/~ >:i.2 2 3 3 4	*/~ >:i.2 1 2 2 4

【「%:」:Square Root(平方根)・Root(:累乗根)】

]r=:%: 2 4 9 1.41421 2 3	***: r** **2 4 9**	「%:」は「*:」の逆演算である。	
%: 0j1	**%: 0j_1**	%: 1j1	%: 4 0j4

0.707107j0.707107	**0.707107j_0.707107**	1.09868j0.45509	2 1.41421j1.41421
3 %: 8 27 2 3 （立方根を出力）	0.5 %: 2 3 4 9 （平方値を出力）	_1 %: 4 0.25 （逆数を出力）	_2 %: 4 0.5 （逆数の２倍）

【「,」(Ravel・Append)：「,.」(Ravel Items・Stitch)：「,:」(Itemize・Laminate)】

コンマ(,)という　動詞の　片側形は　右引数を　リストに変換
コンマ(,)という　動詞の　両側形は　ランクを増やさず　左右を接続
コンマにピリ(,.)の　片側形は　右引数を　テーブル化
コンマにピリ(,.)の　両側形は　左右の引数を　横に接続
コンマにコロン(,:)の　片側形は　ランクを１つ　上げたアレイに
コンマにコロン(,:)の　両側形は　ランクを上げた　アレイになる

【(, Ravel),(,. Ravel Items),(,: Itemize)：アレイの変形(片側形)】

]A=:>:i.2 2 1 2 3 4	$ A 2 2 $ B=:>:i.2 2 2 2 2 2	]a=: ,A 1 2 3 4]b=:,B=:>:i.2 2 2 1 2 3 4 5 6 7 8	$&$ a 1 $&$ b 1
「,(Ravel)」の片側形は、アトム、リスト、テーブル等全てが「リスト化」される。			
]c=:,.1 2 1 2	**]C=:,.>:i.2** **2** **1 2** **3 4**	]D=:,. >:i.2 2 2 1 2 3 4 5 6 7 8	$L:0 c;C;D 2 2 2 1 2 4
「,.(Ravel Items)」の片側形は、右引数の全てのアレイを「テーブル化」する。			
$ c=:1 2 2	$,:c=:1 2 1 2	$,:A=:>:i.2 2 1 2 2	$,:B=:>:i.2 2 2 1 2 2 2
「,:(Itemize)」という動詞の片側形は、ランクを１つ上げる。			

【(, Append),(,. Stitch),(,: Laminate)：アレイの接続(両側形)】

```
A=:>:i.2 3

b=:7 8 9
```

```
   1 2 3 , 7 8 9

1 2 3 7 8 9
```

```
   A , b

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

```
   1,.b

1 7
1 8
1 9
```

```
   (|:A),.b

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

「,(Append)」という動詞の両側形は、左右の引数の最大ランクの方に接続する。

「,.(Stitch)」の両側形はランクが増すことも、変わらないこともある。

```
   ]a=:1,:2


1
2
```

```
   ]A=:1 2 3,:4 5 6

1 2 3
4 5 6
```

```
   ]B=:A,:7 8

1 2 3
4 5 6

7 8 0
0 0 0
```

```
   $ a

2 1
```

```
   $ A

2 3
```

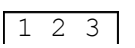
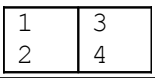
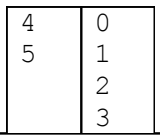
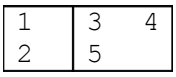
```
   $ B

2 2 3
```

「,:(Laminate)」による接続は、ランクが必ず１つ上がる。「形」が不揃いの箇所には「0」が付加される。

ボックス(<)で　囲めば全てが　アトムに変身　オープン(>使って　蘇生する ボックスで　囲み連結　セミコロン(;)片側形なら　リストに変身！ 小にピリ(<.)　片側形は　切り捨てる 大にピリ(>.)なら切り上げる 数値から　１をマイナス　小にコロン(<:)　大にコロン(>:)は　１を加え			

る 小にピリ(<.)　両側形は　小さいほう 大にピリ(>.)なら大きいほう 小にコロン(<:) 大にコロン(>:)の両側形は　等号含む論理演算			

【「<」Box（ボックス）:「>」Open（オープン）「;」Raze（ほぐし）・Link（結合）】

]A=:<1 2 3  1 2 3	> A 1 2 3	($;#)A  ([$A]は[#A]は「1」)	「ボックス」で囲んだものは「アトム」である。
]B=:1 2 ; 3 4  1 3 2 4	> B 1 2 3 4	; B 1 2 3 4	「; (raze)」で開いた結果は「リスト」である。
4 5 ; i.2 2  4 0 5 1 2 3	]C=:1 2 ; 3 4 5  ; C 1 2 3 4 5	> C 1 2 0 3 4 5	「> open」で開いたとき、形が不揃いなら「0」を表示する。

【「<.」Floor（切捨て）:「>.」Ceilng（切上げ）:片側形】

<. 3.14 3	>. 3.14 4	sgn 3.54 4	round 3.45 3
「sgn=:<.@+&0.5」や「round=:".@(0&":)」は「四捨五入」する関数である。			

【「<:」(Decrement):「>」(Increment):片側形】

<: 3 3.14 2 2.14	(-&1)3 3.14 2 2.14		「<:」は1減
>: 3 3.14 4 4.14	(+&1)3 3.14 4 4.14	1&+ 3 3.14 4 4.14	「>:」は1増

【「<:」(Less of)：「>:」(Large of)：両側関数】

```
   3  <. 3.14
3
```

```
   <./ 3 3.14 4
3
```

```
   3  >. 3.14
3.14
```

```
   >./ 3 3.14 4
4
```

「<.」は小さい方、「>.」は大きい方の要素を出力する。

【「<:」(Less or Equal)：「>」(Large or Equal)：両側関数(論理演算)】

```
   3 <: 3.14
1
```

```
   3 <: 3
1
```

```
   3< 3
0
```

```
   3 >: 3.14
0
```

```
   3>:3
1
```

```
   3 > 3
0
```

「<:」や「>」の両側形は等号付不等号の

論理演算である。

データを　分類するなら　イコール(=)の　片側形を　使えばよい

【100人の生徒の大小順に並べた成績の(架空の)データを「TEST」という変数に入力】

```
   5 20 $ TEST=:(F=:(|.F),20,F=:17 12 7 3 1)#40++:i.11

40 42 42 42 44 44 44 44 44 44 44 46 46 46 46 46 46 46 46 46
46 46 46 48 48 48 48 48 48 48 48 48 48 48 48 48 48 48 48 48
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50
52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 54 54 54
54 54 54 54 54 54 54 54 54 56 56 56 56 56 56 56 58 58 58 60
```

```
sdev=:[:%:var=:[:mean*:@dev [ dev=:-mean=:+/%#
```

```
   mean TEST
50
```

```
   var TEST
16.08
```

```
   sdev TEST
4.00999
```

平均は50、分散はほぼ16、したがって標準偏差はほぼ 4 である。

【「=」の片側形(Self-clasify)は、データの分類にチョー便利なのである！】

```
   T=:3 5 3 2 4 6 4 5

   ]U=:(val=:[:~./:~)T
2 3 4 5 6
   ]F=:(freq=:[:+/"1 val=/])T
1 2 2 2 1
   classify=:val,:freq
```

```
   = T
1 0 1 0 0 0 0
0
0 1 0 0 0 0 0
1
0 0 0 1 0 0 0
0
0 0 0 0 1 0 1
```

```
   U =/ T
0 0 0 1 0 0 0 0
1 0 1 0 0 0 0 0
0 0 0 0 1 0 1 0
0 1 0 0 0 0 0 1
0 0 0 0 0 1 0 0
```

	0 0 0 0 0 0 1 0 0	

「/:~y」は昇順への並べ替え：「~.y」は重複要素の排除

U,:F	classify T	]TC=:classify TEST
2 3 4 5 6 1 2 2 2 1	2 3 4 5 6 1 2 2 2 1	40 42 44 46 48 50 52 54 56 58 60 1 3 7 12 17 20 17 12 7 3 1

【分類されたデータの平均と分散】

]M=:(+/U*F)%N=:+/F	]V=:(+/F**:U-M)%N	mean T	var T
4	1.5	4	1.5

meanc=:3 :'(+/*/y)%+/{:y' varc=:3 :'mean({:y)#*:({.y)-meancy'	分類されたデータの平均と分散を出力する関数（引数には度数分布を入力する）

meanc U,:F	varc U,:F	meanc TC	varc TC	分類しない場合の演算結果と一致する。
4	1.5	50	16.08	

【累積度数法による平均と分散の計算（等間隔で分類されている場合に限る！）】

rsk1=:[:+/sub=:+/\&.|.	rsk2=:[:+/[:}.sub^:2	第１(2)累積度数の和
meanr=:4 :'({.x)+({:x)*<:(rsk1%+/)y' NB.左引数には最小値と級間隔の値を入力 varr=:4 :'(*:x)*((+:rsk2 y)+s-(*:s=.rsk1 y)%n)%n=.+/y' （分散の場合は左引数に級間隔だけを入力すればよい）		

```
   sub F=:1 3 7 12 17 20 17 12 7 3 1
100 99 96 89 77 60 40 23 11 4 1
   }.sub^:2 F
500 401 305 216 139 79 39 16 5 1
```

度数(F)	第１累積度数	第２累積度数
1	100	—
3	99	500
7	96	401
12	89	305
17	77	216
20	60	139
17	40	79
12	23	39
7	11	16
3	4	5
1	1	1

```
   ]S=:rsk1 F
600
```
（第１累積度数の和）

```
   ]T=:rsk2 F
1701
```
（第２累積度数の和）

```
   40 + 2 * <:S %+/F
50
```

```
   40 2 meanr F
50
```

```
   (*:2)*((+:T)+S-(*:S)%n)%n=.+/F
16.08
```

```
   2 varr F
16.08
```

「meanc」や「varc」の結果と一致する。

合計	S=600	T=1701

【接続詞　右にこだわる　接着剤】

片側の　動詞を順に　結ぶのが　アンド(&)やアット(@ @:)の　接続詞

アンダー(&.)で　２つの動詞を　連結すれば　逆演算が　付加される

動詞と名詞を　アンド(&)で結べば　新たな動詞を　作り出す("@"は不可)

複数の　動詞を交互に　連結するのは　タイ(` tie)と呼ばれる　接続詞

【 (「&」And) : (「@」Atop)】

《単項の場合》		《二項の場合》		
「u&v y」「u@v y」は「u{v(y)}」と同じ u も v も単項動詞		[x(u&v)y] は [v(x)uv(y)]v が単項 u は二項 [x(u@v)y] は [u{xvy}]v が二項で u は単項		
*:&+: 2 3 4 16 36 64	*:@+: 2 3 4 16 36 64	2 *&+: 3 24	2 +:@* 3 12	2 +:@+ 3 10

【 (「&.」Under) . (「&:」Appose) . (「@.」Agenda) . (「@:」At)】

*:&.+: 2 3 4 8 18 32	-:&*:&+: 2 3 4 8 18 32	「u&.v」は「u&v」を演算した後で、さらに「v の逆演算」が実行される。

+:`-:@.(2&|&<.)>:i.4 0.5 4 1.5 8	10 +:@- 4 12	10 +:@:- 4 12	10 +:&- 4 domain error

2&* 2 3 4 4 6 8	(*&2)2 3 4 4 6 8(カッコは必要)	動詞と名詞をアンド(&)で結んで新たな動詞 左の例では「+:」という動詞と同じになる。

p=:'abcd';'efgh' Q=:'ABCD';'EFGH' p,&>Q abcdABCD efghEFGH	p,&:>Q abcd efgh ABCD EFGH	(>p),.>Q abcdABCD efghEFGH	(>p),>Q abcd efgh ABCD EFGH	,&>b.0 0 0 0 ,&:>b.0 _ _ _

【「 `」 (Tie) 】

+`*/i.6 29	0+1*2+3*4+5 29	「u`v/」は引数の間に交互に挿入する。
(+:`-:`:0)3 2 6 6 4 12	(+:,:-:)3 2 6	「u`v`:0」は全ての動詞を演算する。

1.5 1 3	6 4 12 1.5 1 3		
(+`*`:3)3 2 6 15	(+`*/) 3 2 6 15	3 + 2 * 6 15	「u`v`:3」は 「u`v/」と同じに機能
(+`*`:6)3 2 6 4 3 7	(+*)3 2 6 4 3 7	a+*a=:3 2 6 4 3 7	「u`v`:6」は「uv」というフックと同じ演算
(+:`*`-:`:6)3 2 6 9 4 36	(+:*-:)3 2 6 9 4 36	「u`v`w`:6」は「uvw」というフォークと同じ演算	

《複素数に対する特有の演算》

【[*(Signum)」と「%-(Reciprocal)」の片側形]】

* _5 0 2 1 0 1	* 3j4 3j_4 0.6j0.8 0.6j 0.8	「*」の片側形は単位円周上への射影(実数の場合は符号だけ)
% r=:2 0.5 _0.25 0.5 2 _4 % c=:3j4 3j_4 0.12j_0.1 0.12j0.16	1 % r 0.5 2 _4 1 % c 0.12j_0.16 .12j0.16	右引数で与えた数値(複素数もO.K.)の逆数を出力する関数「% y」は「1 % y」と同じである。ここで「%」の両側形は割算である。

【[+(Conjugate)」と「-(Negate)」の片側形]】

+ 0.4 _5 0 0.4 5 0	+ 3j4 3j_4 3j 4 3j4	「+」の片側形は虚数部だけ反対符号に、つまり共役複素数(実数の場合はソノママ)。
+ 0.4 _5 0 0.4 5 0	- 3j4 3j_4 3j 4 3j4	「-」の片側形は 実数部、虚数部共に反対符号にする。

【[+.(Real/Imaginary)」と「*.(Length/Angle)」の片側形]】

+. 0j0.5p1 1 1.5708	*. 0 j1 1 1.5708	「*.」の片側形は、複素数の極座標(絶対値と偏角)を出力する。 (「0.5p1」は[π/2=1.5708]である) 極座標表示では、 逆数の絶対値は元の複素数の絶対値の逆数 偏角は符号が反対になる。
+. 3j4 3 4	*. 3j4 5 0.927295	
*. 3j4 5 0.927295	*.% 3j4 0.2 0.927295	

【「j.」Imazjinary(虚数生成)・Complex(複素数生成)】

]b=:j.a=:1j2 3j4 _2j1 _4j3	]c=:{:@*. a 1.10715 0.927295	]d=:{:@*. b 2.67795 2.49809	0.5p1 ; d-c 1.5708 1.5708 1.5708 (c は a の偏角 d は b の偏角)
複素平面上での 90 度(π/2=0.5p1)の回転 [2j1=j.1j2]の偏角と[1j2]の偏角の差は[π/2]			
3 j._1 4 3j 1 3j4	1 3 j._1 4 1j 1 3j4	j./ 1 1 1j1	j./ 0 _1 0j 1

【「r.」Angle(単位複素数)・Poler(極座標表示)】

r. 0 1p1 1 0j1 _1	r. 05p1 1.5p1 1 0j 1	引数が偏角の単位複素数 [0.5p1],[1p1],[.5p1]はそれぞれ [π/2], [π], [3π/2]
2 r. 0.5p1 0j2	*.2 r.0.5p1 2 1.5708	「r.」の両側形は片側形の結果に左引数倍 (「x r.y」は「x*r.y」の演算結果と同じ)

【複素数演算と２次元平面上の一次変換】

複素数演算	一次変換の行列	行列計算	コメント
(*&1) 3j4 3j4	]E=:2 2 $ 1 0 0 1 1 0 0 1	E(mp=:+/ .*)3 4 3 4	恒等変換
(*&2)3j4 6j8	]E2=:2*E 2 0 0 2	E2 mp 3 4 6 8	相似変換

+ 3j4 3j_4	]X=:2 2 $ 1 0 0 _1 1 0 0 1	X mp 3 4 3 _4	実軸(横軸)に関して対称変換する。
-&+ 3j4 _3j4	]Y=:-X _1 0 0 1	Y mp 3 4 _3 4	虚軸(縦軸)に関して対称変換する。
|.&.+. 3j4 4j3	]LX=:-.E 0 1 1 0	LX mp 3 4 4 3	直線「$y = x$」に関して対称変換する。
+&j. 3j4 _4j_3	]LY=:-LX 0 _1 1 0	LY mp 3 4 _4 _3	直線「$y = -x$」に関して対称変換する。

j. 3j4 _4j3	]R90=:2 2 $ 0 _1 1 0 0 _1 1 0	R90 mp 3 4 _4 3	(a,b)を平面上で90度回転する。
(j.^:2)3j4 _3j_4	]R180=:-E _1 0 0 1	R180 mp 3 4 3 4	(a,b)を平面上で180度回転する。
(j.^:_1)3j4 4j 3	]R_90=:-R90 0 1	R_90 mp 3 4	(a,b)を平面上で時計回りに90度回転。

	1 0	4 3	
(+/,{.)&.+.3j4 7j3	]P=:2 2 $ 1 1 1 0 1 1 1 0	P mp 3 4 7 3	(a,b) を $(a+b,b)$ に変換

【３点 $A=(4,3)$, $B=(1,0)$, $C=(7,0)$ で与えられる三角形の重心の位置を求める問題】

T=:4j3;1;7 gpoint=:+/@:>%3:	gpoint T 4j1	△ABCの位置を複素数で表示して「T」という変数に入力。「gpoint」は重心を求める関数

【７点一致の問題：任意の四角形ＡＢＣＤに対して、次の７つの点が全て一致する】

① 辺ＡＢの中点と辺ＣＤの中点を結ぶ線分の中点：P１

② 辺ＢＣの中点と辺ＤＡの中点を結ぶ線分の中点：P2

③ 対角線ＡＣの中点と対角線ＢＤの中点を結ぶ線分の中点：P3

④ 三角形ＡＢＤの重心と頂点Ｃとを結ぶ線分を１：３に分ける点：P4

⑤ 三角形ＡＢＣの重心と頂点Ｄとを結ぶ線分を１：３に分ける点：P5

⑥ 三角形ＢＣＤの重心と頂点Ａとを結ぶ線分を１：３に分ける点：P6

⑦ 三角形ＡＣＤの重心と頂点Ｃとを結ぶ線分を１：３に分ける点：P7

'A B C D'=:4j8;6j6;8;0		四角形ＡＢＣＤを複素数値で与える。
mpt=:[:-:[:+/> div4=:[:+/4:%~]*1:,3:	「mpt」は中点を与える関数	]P1=:mpt(mpt A;B);mpt C;D 4.75j3.5
]P2=:mpt(mpt B;C);mpt D;A 4.75j3.5		]P3=:mpt(mpt C;A);mpt B;D 4.75j3.5
]P4=:div4 C,gpoint A;B;D 4.5j3.5		]P5=:div4 D,gpoint A;B;C 4.5j3.5
]P6=:div4 A,gpoint B;C;D 4.5j3.5		]P7=:div4 B,gpoint A;C;D 4.5j3.5

感嘆符(!)　片側形は　階乗よ　両側形は　２項係数　（“!.”や“!:”は 接続詞）

ビックリピリ(!.)は　ハット(^)と数を接続し 両側動詞を　生成する(custemize)

【“! !. ^!.0 ^!.1 ^!.2 ^!. 1 ^|. 2 !:”】

! 3 4 5 6 24 120	! 0.5 1.5 2.5 0.886227 1.32934 3.32335	1 1.5*-:%:1p1 0.886227 1.32934
(bic=:i.@>:!])5 1 5 10 10 5 1 ${}_5C_0,{}_5C_1,{}_5C_2,{}_5C_3,{}_5C_4,{}_5C_5$	bden=:4 :'(k!y)*(x^|.k)*(-.x)^k=.i.1+y' 0.5 bden 4 0.0625 0.25 0.375 0.25 0.0625	

(3 :'y e.~&>/;y' y=:(<'abc'),(<'d'),<'a'	e. y

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

【(「#」Copy：両側形)：(「#.」Base2．Base)：(「#:」Antibase2． Antibase)】

右で与えた　データから　左指定の　個数取り出す　両側コピー(# copy)

シャープ・ピリ(#.)　片側形は　２進数を　10進数の　数値に変換

左で与えた　進数で　右の数値を変換す　シャープ・ピリ(#.)の　両側形

シャープ・コロン(#:)　片側形は　10進数を　２進の数値に　変換す

シャープ・コロン(#:)　両側形は　シャープ・ピリ(#..)の　両側形の逆変換

【「#」(Copy)：両側形】

```
   3 # 1

1 1 1

   0 1 1 # 1 3

5

3 5

   1 2 3 # 1 3

  5

1 3 3 5 5 5

  2 1 3 # 'abc'

aabccc
```

```
   ]A=:>:i.2 3

1 2 3

4 5 6

   1 0 # A

1 2 3

   1 2 # A

1 2 3

4 5 6

4 5 6
```

```
   ]E=:=/~i.3

1 0 0

0 1 0

0 0 1 (単位行列)

   ]B=:>:i.3 3

1 2 3

4 5 6

7 8 9
```

```
   E #"1 B

1

5

9

   (<0 1)&|: B

1 5 9

 (Bの対角要素)

   (<0 1)&|: A

1 5
```

【「#.」(Base2)：片側形】

```
   #.1 0 1

5
```

```
   +/(1 0 1)*2^2 1 0

5
```

２進数で「1 0 1」は10進数では「5」[#.]の片側形は２進数を10進数に変換

```
   bi_10 1 0 1

5
```

```
   biten 1 0 1

5
```

```
bi_10=:3 :'+/y*2^i.-#y'

biten=:[:+/]*2:^[:i.-@#
```

【「#.」(Base)：両側形】

```
         10   #.
d=.1+i.4

1234

   8 #. d

668
```

```
     +/d*(10^3 2 1 0)

1234

   +/d*(8^3 2 1 0)

668
```

10進数で「1 2 3 4」は「1234」

８進数で「1 2 3 4」は「668」

$1\times 8^3 + 2\times 8^2 + 3\times 8^1 + 4\times 8^0$

$= 512 + 128 + 24 + 4 = 668$

```
   eit_10 d

668
```

```
   eiten d

668
```

```
eit_10=:3 :'+/y*8^i.-#y'

eiten=:[:+/]*8:^[:i.-@#
```

【「#:」(Antibase2)：片側形】

]c=:#:3 5 7	#. c	#:b._1	#.b._1
0 1 1 1 0 1 1 1 1	3 5 7	#.	#:
	(#:^:_1)c	「#:(片側形)」は「#.(片側形)」の逆変換	
	3 5 7	「v b. 1」は関数「v」の逆関数を出力する。	

【「#:」(Antibase):両側形】

]c=:4 4#: 5+i.3	4 #. c	]d=:(3$4)#:31+i.3	4 #. d
1 1 1 2 1 3	5 6 7	1 3 3 2 0 0 2 0 1	31 32 33
	c +/ .*4^10 5 6 7		d+/ .*4^i._3 31 32 33

a=:24 60 60 b=:3600 60 1	]s=:a #.2 3 4 7384	$(2\times 60\times 60)+(3\times 60)+4$ $=7200+180+4=7384$
b #: s 2 3 4	+/2 3 4 * b 7384	「2時間3分4秒」は「7384秒」である。 逆に「7384秒」は「2時間3分4秒」

ハット(^)という　動詞の片側形は　指数関数を　出力す
ハット(^)という　動詞の両側形は　右の数だけ　累乗す
ハットピリ(^.)　片側形は　自然対数　両側形は　左を底の対数値
ハットコロン(^:)　反復演算の　接続詞　マイナス１なら　逆演算

【「^」Exponential(指数関数)・Power「x」オイラーの定数等を与える名詞】

^ 0 1 2	1x0 1x1 1x2	(1x1)^ i.3
1 2.71828 7.38906	1 2.71828 7.38906	1 2.71828 7.38906
*:123456789	x:*:123456789	3 :'(1+%y)^y' 200000
1.52416e16	15241578750190521	2.71828

2 3 ^ 3 4	2 3 ^ 2	^&2(2 3)	*: 2 3
8 81	4 9	4 9	4 9

【「^.」Natural-Log（自然対数関数）・Logalithm：「^:」Power】

]t=:1x0 1x1 1x2 1 2.71828 7.38906	^ 0 1 2 1 2.71828 7.38906	
^.t 0 1 2	3 :'1x1^.y' t 0 1 2	「^.y」は(自然)対数関数 「x^.y」は[x]を底とする[y]の対数
]s=:10^i.3 1 10 100	3 :'10^.y' s 0 1 2	
^. 1 2 0 0.693147	o_log s 0 1 2	「o_log=:3 :'10 ^.y'」は常用対数値を出力する関数

>:^:2(0 1 2) 2 3 4	>:>: 0 1 2 2 3 4	>:^:_1 (2 3 4) 1 2 3	<: 2 3 4 1 2 3

被負整数を　瞬時に作る　アイにピリ(i.)　但し始点は　0にご注意(1ではない！)

マイナスの　整数値まで　出力す　iにコロン(i:)は　重宝動詞(但しランク0に作動)

【「i.」(Integers)：片側形】

i.5 0 1 2 3 4	i._5 4 3 2 1 0			「i.」は0から始る自然数列を生成する原始動詞。引数が負の場合には逆順になる。
|.;}.0:`(];$:@<:)@.*5 0 1 2 3 4			1:`(]*$:@<:)@.*5 120	! 5 120
i.2 3 0 1 2 3 4 5	i._2 3 3 4 5 0 1 2	i.2 _3 2 1 0 5 4 3	i._2 _3 5 4 3 2 1 0	アトムでなくリストを入力するとテーブル等の一般アレイを出力する。
「始点」を「0」ではなく「1」にすべきであるとシツコク主張するユーザーもいる。 (J言語の前身である「APL」では、始点を「0」と「1」のいずれにするかが選択できた！)				

【「i:」(Integers)：片側形】

i:3 _3 _2 _1 0 1 2 3	i:_3 3 2 1 0 _1 _2 3	「i:y」は0を挟んでy>0なら(-y)から(y)までの整数列を生成(y<0なら逆順)。
icolonr=:3 :'|.^:(-:1-*y)(-|y)+i.>:+:|y'		「i:」と同じ実数に対する関数

]a=.i.1+| 3 0 1 2 3	|. a 3 2 1 0	-@}. a _1 _2 _3	(|.,-@}.) a 3 2 1 0 _1 _2 3
|.^:1(|.,-@}.)a 3 2 1 0 1 2 3	|.^:0(|.,-@}.)a 3 2 1 0 1 2 3		
icoronr 3 3 2 1 0 1 2 3	icoronr _3 3 2 1 0 1 2 3		

「i.」や「i:」は整数値でない引数に対しては” domain error”となる。また「i:」はランク 0 の引数に対してのみ作動する。		
i: 3j4 _3 _1.5 0 1.5 3	i: _3j4 3 1.5 0 _1.5 3	「b」が正整数の「ajb」という複素数に対しては、[-a]から[a]までで, [2a/b]という間隔の数列を出力する。a が負のときは逆順になる。
i: 1.5j4 _1.5 _0.75 0 0.75 1.5	i: 1.5j3 _1.5 _0.5 0.5 1.5	
icolonc 3j4 _3 _1.5 0 1.5 3	icolonc _3j4 3 1.5 0 _1.5 3	icolonc=:3 :0 c=.b%~2*a ['a b'=.+.y (-a)+c*i.1+2*a%c) NB.「i:」と同じ働きの複素数への関数。
icolonc 1.5j4 _1.5 _0.75 0 0.75 1.5	icolonc 1.5j3 _1.5 _0.5 0.5 1.5	
icolon=:3 :'if.0={:+.y do.icolonr y else.icolonc y end.'		
icolon 3 _3 _2 _1 0 1 2 3	icolon 3j4 _3 _1.5 0 1.5 3	「icolon」は「i:」の片側形とソックリ同じで、実数、複素数とも何れでもオーケー

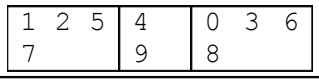
【(「i.」Index of) . (「i:」Index of Last):両側形】

'abcdef'i.'aec' 6 0 4 2 'abcae'i.'acge' 0 2 5 4	'abcdef'index'aec' 6 0 4 2 'abcae'index'acge' 0 2 5 4	「i.」の両側形は 左の要素のインデックスを　右引数のリストに与える。 左の要素にない空欄や g にはインデックス外の数値 (6) や (5) が表示される。
97 66 99 { a. aBc	a. i. 'a b' 97 32 98	「a.」には 256 個の文字やキャラクターが入力されている。
‘A’ -: 65{ a. 1	‘ ‘-: 32{a. 1	「a.」のインデックス 65 には[A]という文字、インデックス 32 にはスペース
83 85 90 85 75 73 32 71 105 105 116 105 114 111 { a. SUZUKI Giitiro		小生の「姓名」が出力された。
a. i. 'SUZUKI Giitiro' 83 85 90 85 75 73 32 71 105 105 116 105 114 111		

index=:4 :',t{.@#"1 i.{:$t=.y=/x,y-.x'		indexb=:4 :',t{:@#"1 i.{:$t=.y=/x,y-.x'	
'abcae'i:'acge' 3 2 5 4	'abcae'indexb'acge' 3 2 5 4	「i:」の両側形も「i.」の結果とほぼ同じ　ただインデックスは後ろから	
a. i: 'A a' 65 32 97	a.index'A a' 65 32 97	a.indexb'A a' 65 32 97	a.indexb'A a' 65 32 97

【「I.」Indices】

a=:0 0 1 0 2 0 index=:#i.@#	a #:(i.@#)a 2 4 4	index a 2 4 4	I. a 2 4 4
]c=:?.10$20 6 15 19 12 14 19 0 17 0 14	10 I.@:< c 1 2 3 4 5 7 9	10(<index])x 1 2 3 4 5 7 9	
10 I.@:> c 0 6 8	10 ([:I.>) c 0 6 8	10 (>index]) c 0 6 8	10 ((<;>)#L:0 i.@#@]) c 1 2 3 4 5 7 9 | 0 6 8
14 I.@:< c	14 I.@:= c	14 I.@:> c	14 ((<;=;>)#L:0 i.@#@]) c

1 2 5 7	4 9	0 3 6 8	
C-:|."1A 1	D-:|.A 1	E-:|.|."1A 1	「|.」や「|."1」はそれぞれ行や列に関する転置で、「-:」はソックリ同じなら「1」

【「|」(Magnitude・Residue)・「|.」(Reverse・Rotate)・「|:」(Transpose)】

割算の　余り求める　棒(|)一本　片側形なら　絶対値

棒にピリ(|.)　片側形なら　アイテムの　順序をそっくり　逆にする
棒ピリ(|.)の　両側形は　左の数だけ　右に回転(rotate)　負なら左へ
棒にコロン(|:)　片側形なら アレイの軸の　順序をソックリ　入れ替える
棒にコロン(|:)　両側形は　左指定の　軸を 0 軸に　転置(transpose)する
左にボックスの　データを入力すれば　対角要素を　出力する

【「|」(Magnitude・Residue)】

```
   | 1 _2 3 _4
1 2 3 4
```

```
   | 1j1 3j4
1.41421 5
```

```
res=:4 :'x*u-<.u=.y%x'
res1=:[*%~-[:<.%~
```

```
   3 | i.6
0 1 2 0 1 2
```

```
   1j2 | 3j4
1
```

```
   3 res i.6
0 1 2 0 1 2
```

```
   1j2 res 3j4
 1j1
```

```
   1j2 res1 3j4
1j3.33067e 16
```

```
   (2*1j2)+1
3j4
```

```
(2j_1*1j2)+_1j1
3j4
```

いずれを剰余と考えるべきか？

```
   1j2 | 2j2 3j3 3j4
1 _1 1
```

```
     1j2 res 2j2 3j3 3j4
_1j1   _1j1.11022e_16
 1j1
```

```
     1j2 res 2j2 3j3 3j4
_1j1   _1j1.11022e_16
 1j1
```

```
   (1*1j2)+1
2j2
```

```
    (2j_1*1j2)+_1
3j3
```

```
   (2*1j2)+1
3j4
```

```
    1j2 res1 2j2 3j3 3j4
_1j1   _1j1.11022e_16
 1j1
```

【「|.」(Reverse・Rotate)】

```
   |.d=:1 2 3 4 5
5 4 3 2 1
```

```
   |. i.2 3
3 4 5
0 1 2
```

```
   |."1 i.2 3
2 1 0
5 4 3
```

「|.」の片側形はアイテムの順序の逆転

```
   1 |.d
2 3 4 5 1
```

```
   2 |.d
3 4 5 1 2
```

```
   1 |.^:2 d
3 4 5 1 2
```

右への回転

```
   _1 |. D
5 1 2 3 4
```

```
   _2 |. d
4 5 1 2 3
```

```
   _1 |.^:2 d
4 5 1 2 3
```

左への回転

【「|:」(Transpose)】

```
    ]M=:|:>:i.3
```

```
    ]m=:2
```

```
   |:m
```

```
   1 |: M
```

3 1 4 7 2 5 8 3 6 9	3$'abcdef' abc def	 ad be cf	 1 4 7 2 5 8 3 6 9
1 |: m abc def	0 1|:m abc def	(<0 1)|:m Ae	(<0 1)|:M 1 5 9 （対角要素の出力）

データに　プラス・スラッシュ(+/)　合計算

スラッシュ・ピリ(/.)　右引数の テーブルを　逐一斜めの　対角要素 (oblique)

データを 昇順にする グレードアップ(/:~)　グレードダウン(\:~)は　降順に

【(「v/y」Insert・Table) : (「+/ .*」Matrix Prodact) 】

+/ 4 2 3 9	*/ 4 2 3 24	=/~ i.3 1 0 0 0 1 0 0 0 1	*/~>:i.3 1 2 3 2 4 6 3 6 9
-/ 4 2 3 5	%/ 4 2 3 6		
]a=:3+/\ t=:>:i.5 6 9 12	a % 3 2 3 4	3 mave t 2 3 4	「mave=:+/\%[」は 移動平均を出力する

【「v\y」Prefix 】

prefix=:3 :'(>:i.#y)<@{."0 1 y'	prefix_t=:>:@i.@#<@{."0 1]
<\ t=:>:i.5 1 | 1 2 | 1 2 3 | 1 2 3 4 | 1 2 3 4 5 ‘’	prefix t 1 | 1 2 | 1 2 3 | 1 2 3 4 | 1 2 3 4 5
「<\」の片側形は１から「#y」まで前から累加。	

【「x v\y」Infix 】

cross=:4 :'<"1 x{."1(i.>:(#y)-x)}."0 1 y'	
dis=:4 :'(}:t),(s-x*r)}.L:0{:t=.<"1((r=.>.(s=.#y)%x),x)$y'	
sub0=:](],#@[-+/@])[$~[:<:@>.[%~#@]	sub=:[:sub1[sub0]
sub1=:(('',{.);<"1@(}.,.}:))@(+/\)	dis t=:[:-./L:0 sub{."1"0 1

L:0]

infix=:4 :'if.x>0 do.x cross y else.(-x)dis_t y end.' （「<\」の両側形と同じ)

2 <\ t=:>:i.5

1 2	2 3	3 4	4 5

3

<\ t

1 2 3	2 3 4	3 4 5

2 cross t

1 2	2 3	3 4	4 5

3 cross t

1 2 3	2 3 4	3 4 5

_2 <\ t

1 2	3 4	5

_3

<\ t

1 2 3	4 5

2 sub t

2	4 2	5 4

3 sub t

3	5 3

‘’

2 dis_t t

1 2	3 4	5

3

dis_t t

1 2 3	4 5

2 infix t

1 2	2 3	3 4	4 5

_2 infix t

1 2	3 4	5

「<\(infix)」の両側形は左引数が正(負)なら重複を許し(許さず)個数分逐次出力する。

【「v/:y」Oblique：片側形】

```
   ]A=:>:i.3 4

1  2  3  4
5  6  7  8
9 10 11 12
```

```
   </. A
```

1	2 5	3 6 9	4 7 10	8 11	1 2

```
</. |:A
```

1	5 2	9 6 3	10 7 4	11 8	1 2

「</.(oblique)」は右引数の テーブルを逐一斜めの対角要素を出力する。

```
half=:3          :'<"1@([:<"1],.|.)@i."0>:i.<./$y'
box=:3           :'<"1(-i.-<./d)+/<:d=.$y'
oblique=:diag{L:1] [ diag=:3 :'t,(box y)+"0 L:0|.t=.half y'
```

half A

0 0	0 1	1 0	0 2	1 1	2 0

box A

0 1	1 2	2 3

(diag{ L:1]) A

1	2 5	3 6 9	4 7 10	8 11	12

diag A

0 0	0 1	1 0	0 2	1 1	2 0	0 3	1 2	2 1	1 3	2 2	2 3

【「x v/:y」Key：:両側形】

```
   b=:1 2 3 1 3 2 1

   b          </.
A=:'abcdefg'
```

adg	bf	ce

```
   ]a=:=b

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

```
   a <@# A
```

adg	bf	ce

```
key=:4 :'(=x)<@# y'

key_t=:=@[<@#]

   b key A
```

adg	bf	ce

【「v\.y」Sufix】

```
   <\.t=.>:i.5
```

1 2 3 4 5	2 3 4 5	3 4 5	4 5	5

(「v\.y」sufix)は１から「#y」まで前から減少させる。

【「x v\.y」Outfix】

```
   3 <\. t=:1 2
3 4 5
```

4 5	1 5	1 2

```
3 outfix t
```

4 5	1 5	1 2

```
   2 <\. t
```

3 4 5	1 4 5	1 2 5	1 2 3

```
2 outfix .t
```

3 4 5	1 4 5	1 2 5	1 2 3

```
drop=:4 :'(<<<x){y'

outfix=:4 :'(x<\i.#y)drop
L:0 y'
```

【(「/.」Grade up・Sort) ; (「\.」Grade down・Sort)】

```
up=:/:

down=:\:
```

```
   4!:0<'up
'

3
```

```
   4!:0<'dow
n'

3
```

「/:」「\:」は何れも動詞である。

```
   ]a=:/:c=:3
1 2

1 2 0
```

```
      a
{ c

1 2 3
```

```
   c /:
c

1 2 3
```

```
      /:~
c

1 2 3
```

「/:」はアレイの昇順のインデクス

「\:」はアレイの降順インデクス

両側形は昇順・降順のソート

```
   ]b=:\: c

0 2 1
```

```
      b
{ c

3 2 1
```

```
   c \:
c

3 2 1
```

```
      \:~
c

3 2 1
```

【「v/:y」Grade up「x v/:y」Sort ; 「v\:y」Grade down「x v\::y」Sort】

```
   <./ a=:4 2 3 6

2
```

```
   (-.<./)a

4 3 6
```

```
      (-.<./)^:2
a

4 6
```

```
      (-.<./)^:3
a

6
```

```
up=:4 :'<./(-.<./)^:x y'

down=:4   :'>./(-.>./)^:x
y'
```

```
   (i.4)up"0 1 a

2 3 4 6
```

```
   (i.4)down"0 1 a

6 4 3 2
```

```
index_u=:(]=up)#[:i.#@]

index_d=:(]=down)#[:i.#@]
```

```
   ]s=:,(i.4)index_u"0
1 a

1 2 0 3
```

```
   ,(i.4)index_d"0
1 a

3 0 2 1
```

```
grade_u=:[:,i.@#index_u"0 1]
```

```
   ]u=:grade_u
a
```

```
   ]d=:grade_d
a
```

grade d=:[:,i.@#index d"0 1]		1 2 0 3	3 0 2 1
grade_up=:3 :'(i.#y)up"0 1 y' grade_down=:3 :'(i.#y)down"0 1 y'		grade_up a 2 3 4 6	grade_down a 6 4 3 2
/:~ a 2 3 4 6	\:~ a 6 4 3 2	u { a 2 3 4 6	d { a 6 4 3 2

【(「".」Do・Number):(「":」Defort Format・Format)】

1+":2 domain error （数値と文字の足算はエラーになる）	1+".":2 3 （数値化してからなら演算可能）	4j2 ": 3.14159 3.14 5j3 ": 3.14159 3.142	「":」は文字化で、両側形は書式関数。 「".」は数値化
a=:'1+2+3'	". a 6	「".(do)」は文字で記述された演算内容を実行する。	
]b=:'1 2 3','4 5',:'' 1 2 3 4 5 $ b 3 5	8 ". b 1 2 3 4 5 8 8 8 8	1j1 ". b 1 2 3 4 5 1j1 1j1 1j1 1j1	「".」の両側形は隙間に左引数で与えた数値を挿入する。
]d=:3+i.5 3 4 5 6 7	". '5*d' 15 20 25 30 35	「5*d」という演算結果を出力する。	

ボックスで　与えた要素の　組合せ　片側動詞の　カタログ({ Catalogue)なり
中カッコ({}　左で与えた　インデクスの　アイテムを取る　両側関数({ From}
カッコ閉じ()}　左で与えた　インデクスの　アイテム修正　両側関数
修正値と　インデクスを左に　入力すれば　右引数の値を　修正す() Amend)

【「{」{ Catalogue：片側形】

{ 0 1;0 1	{'a';'bc';'def'	{3

0 0 0 1 1 1 0 1	abd abe abf acd ace acf	3 アトムに対しては“ボックス(<)”と同じ

```
]Card=:'CDHS';'23456789TJQKA'

CDHS 23456789TJQKA   { Card
```

C2	C3	C4	C5	C6	C7	C8	C9	CT	CJ	CQ	CK	CA
D2	D3	D4	D5	D6	D7	D8	D9	DT	DJ	DQ	DK	DA
H2	H3	H4	H5	H6	H7	H8	H9	HT	HJ	HQ	HK	HA
S2	S3	S4	S5	S6	S7	S8	S9	ST	SJ	SQ	SK	SA

《52 枚のカード(俗称トランプ)》

【「{」From：両側形】

{ 'abcde' abcde	{"0 'abcde' a b c d e	{ L=:>:i.2 3 1 2 3 4 5 6	]T=:>:i.3 4 1 2 3 4 5 6 7 8 9 10 11 12
1 3 { 'abcde' bd	0 { T 1 2 3 4 0 {"1 T 1 5 9	0 2 { T 1 2 3 4 9 10 11 12	<1 2;2 3 1 2 2 3 (二重ボックス) (<1 2;2 3){ T 7 8 11 12
(h=:1 1 2 0)} T 5 6 11 4 box =:[:<"1],.i.@#	]k=:box h 1 1 2 0 1 1 2 3 k { T 5 6 11 4		

【「{.」Head・Take】

]K=:>:i.3 1 2 3	{. K 1	1{. K 1	2{.K 1 2
_1{.K=:>:i.3	_2{. K	「{.」の片側形は先頭要素の取り。両側形は正(負)なら先頭(末尾)から個数分の取り。	

3	2 3		

【「{:」Tail】

{:K=:>:i.3 3	2 }. K 3	(([:<:#)}.]) K 3	

【「)」Item Amend . Amend】

]A=:2 4$'abcdefgh' abcd efgh	1 0 1 0 } A ebgd	1 0 1 1 } A ebgh	
]B=:'abcde' abcde	'BD'(1 3)}B aBcDe	B abcde	]B=:'BD'(1 3)}B aBcDe
J言語では、訂正したものを定義し直さないと変更されない点に注意。			

【「).」Behead.Drop ; 「}:」Curtail】

]K=:1+i.3 1 2 3	}. K 2 3		
1 }.K 2 3	2 }.K 3		
_1}.K 1 2	_2}.K 1	}: K 1 2	

【乱数の生成 (? ; Roll)と(?. ; Deal)】

? 10$10 8 3 8 1 2 8 0 0 2 1	?. 8$10 6 5 9 2 4 9 0 7	?. 8$10 6 5 9 2 4 9 0 7	「?」の片側形は重複を許さぬ乱数の生成。「?.」はシードを固定
10 ? 10 5 9 6 0 7 3 4 8 2 1	8 ?. 10 6 9 1 4 0 2 3 8	8 ?. 10 6 9 1 4 0 2 3 8	「?」の両側形は重複を許した乱数の生成。「?.」はシードを固定
? 2 8 $ 10	?. 2 8 $ 10	?. 2 8 $ 10	0 から 9 までの重複乱数を 2×8 の形で生成している。「?.」を使えばシードを固定できる。

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

両側動詞に ウエーブ(~)つけりゃ 右引数を左にも 左右に数値がある場合には左右の引数を交換すウエーブ・ピリ(~.nub)の 片側形は重複要素を 排除する ウエーブ・コロン(~:)の 片側形は ダブリの位置に“0”を与える ウエーブ・コロン(~:)の 両側形は 各要素毎の 不一致に“1” マイナス・コロン(-:)の 両側形は 引数マ				

トメテ　一致に “1”(match)				

【「~」Refrex・Passive Evoke】

+/~ a=:1+i.3	a +/ a	*/~ a	a */ a
2 3 4 3 4 5 4 5 6	2 3 4 3 4 5 4 5 6	1 2 3 2 4 6 3 6 9	1 2 3 2 4 6 3 6 9
1 2 3 -~ 5	5 - 1 2 3	5 %~ i.5	(i.5) % 5
4 3 2	4 3 2	0 0.2 0.4 0.6 0.8	0 0.2 0.4 0.6 0.8

【(「~.」Nub/)・(「~:」Nub Sieve/Not-Equal)】

a =: 1 2 1 3 3 2 1	~. a	~: a	nub_s a	(~:a) # a
nub_s=:[:+/[:</\"1=	1 2 3	1 1 0 1 0 0 0	1 1 0 1 0 0 0	1 2 3

「~.nub」の片側形は重複要素の排除で、「~:nub sieve」は重複の箇所に「0」を与える。		
(1 2)~:2 1 1 1	(1 2)~:1 2 0 0	「~:」の両側形は各要素毎の不一致に「1」 そうでなければ「0」を与える論理演算
(1 2)-:2 1 0	(1 2)-:1 2 1	「-:」の両側形は引数マトメテ一致に「1」

セイム(same [,])は　左右のいずれかを　出力させる　便利な動詞

キャップ([:)はなんとも　不思議な動詞　何もしないで　フォークを作る

【[,](Same)・[:(Cap)】

2 3 [4 5	2 3] 4 5	(+:2 3) [4 + 5	(+:2 3)] 4 + 5

2 3	4 5	4 6	9
([:*:+:)2 3 4 16 36 64	([:>:+:)2 3 4 5 7 9	*:&+: 2 3 4 16 36 64	>:&+: 2 3 4 5 7 9
2(+*-)1 3	]a=:(2+1)*(2-1) 3	2([:>:+*-)1 4	>: 4

(abs=:| :[:)_1 2 3 1 2 3	2(res=:[: :|)1 2 3 1 0 1	キャップは　演算結果に　関係せず

【論理演算の概要】

マイナスコロン(-:)の　両側形は　「形」まで含めて　一致(1)か否(0)か(論理演算)

小(<)や大(<)の　両側形は　真なら「1」で偽なら「0」の　論理演算

小にコロン(<:)や　大にコロン(>:)は　イコール含む不等式(≦,≧)の　論理演算

【(「+.」Or)・(「+;」Not-Or)・(「*.」And)・(「*:」Not-And):両側形】

a=:0　0 1 1 b=:0　1 0 1	a +.b 0 1 1 1	a +:b 1 0 0 0	a *.b 0 0 0 1	a *:b 1 1 1 0	基本的な４つの論理演算 (論理和と論理積)

【論理和(O r +.):ブール代数】

a +.b 0 1 1 1	a(0:<+) b 0 1 1 1	a(0:`1:@.*@+)b 0 1 1 1	左右の引数が共に「0」のときだけ「0」を出力し、そうでなければ「1」を出力する。

【否定論理和(Not-Or +:):ブール代数】

a +:b 1 0 0 0	a(0:=+) b 1 0 0 0	a(1:`0:@.*@+)b 1 0 0 0	左右の引数が共に「0」のときだけ「1」を出力し、そうでなければ「0」を出力する。

【論理積(And *.):ブール代数】

a *.b 0 0 0 1	a(1:=*) b 0 0 0 1	a(0:`1:@.*@*)b 0 0 0 1	左右の引数が共に「1」のときだけ「1」を出力し、そうでなければ「0」を出力する。

【否定論理積(Not-And *:):ブール代数】

a *:b 1 1 1 0	a(0:=*) b 1 1 1 0	a(1:`0:@.*@*)b 1 1 1 0	左右の引数が共に「1」のときだけ「0」を出力し、そうでなければ「1」を出力する。

【(「-:」Match) : (「=」(Equal) : (「~:」Not equal) : 論理演算】

20 30 40 = 40 30 20 0 1 0	20 30 40 ~: 40 30 20 1 0 1	「=」は等しければ1、等しくなければ0 「~:」は等しければ0、等しくなければ1
20 30 40 -: 40 30 20 0	20 30 40 -: 20 30 40 1	「-: Match」は「形」まで含めてソックリ同じときだけ「1」
左右の要素毎に等しいときに「0」等しくないときに「1」を出力する。「~:」は‘not equal’		

【(< Less than) . (> Larger than) . (<: Lesser or equal) . (>: Lager or equal)】

3 > 3.14 0	3 < 3.14 1	3 > 3 0	3 < 3 0	これらはいずれも「論理演算」で、「コロン(:)」の付いたほうは「＝」がついてる場合である。
3 >: 3.14 0	3 <: 3.14 1	3 >: 3 1	3 <: 3 1	

【「b.」Boolean.Basic】

+./~ 1 0 1 1 1 0	]S=:7 b./~ 1 0 1 1 1 0	]T=:|:S 1 1 1 0	]t=:|.,T 0 1 1 1	#. t 7
+:/~ 1 0 0 0 0 1	]SN=:8 b./~ 1 0 0 0 0 1	]U=:|:SN 0 0 0 1	]u=:|.,U 1 0 0 0	#. u 8
*./~ 1 0 1 0 0 0	]P=:1 b./~ 1 0 1 0 0 0	]Q=:|:P 1 0 0 0	]q=:|.,Q 0 0 0 1	#. q 1
*:/~ 1 0 0 1 1 1	]PN=:14 b./~ 1 0 0 1 1 1	]R=:|:PN 0 1 1 1	]r=:|.,R 1 1 1 0	#. r 14

boolean=:3 :'#.|.,|:y b.~/~10'		boolean"0 (7 8 1 14) 7 8 1 14	
1 0 +./ 0 1 1 1 0 1	11 b./~1 0 1 1 0 1	1 0 +:/ 0 1 0 0 1 0	4 b./~1 0 0 0 1 0
1 0 *./ 0 1 0 1 0 0	2 b./~1 0 0 1 0 0	1 0 *:/ 0 1 1 0 1 1	13 b./~1 0 1 0 1 1
boolean"0 (11 4 2 13) 11 4 2 13		boolean"0 >:i.15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
^ b._1 ^.	^ b.0 0 0 0	^ b.1 $&1@(}.@$)	

【(「_」Negative Sign/Infinity)・(「_.」Indefinite)・(「:」Indefinity)・(「”_」)】

2 % _	2 % __	_ + _	「_」は∞(無限大)の名詞

0	0		「　」は¯∞（マイナス無限大）
_ ¯ _ .	3 + _. .	2 * _. 0	「_.」は　不定形(indeterminate)の名詞
_: '' 0	(]%_:)2		「_:」は無限大(∞)を出力する動詞
(]+1"_)i.3 1 2 3	(]+1:)i.3 1 2 3		「" _」は数値につけて“動詞化”する副詞。

【複数組の関数を同時に定義するワザ】

```
   stat_reg=:3 :0

regb=:[%.1:,.]
regp=:(1:,.])+/ .*regb
regq=:[:+/[:*:[-regp
regcd=:100"_*1:-regq%[:+/[:*:(]-+/%#)@[
mat=:[:%.(|:+/ .*])@(1:,.])
resvar=:regq%[:-/[:$1:,.]
regt=:regb%[:%:resvar*[:(<1 0)&|:mat@]
mll=:>:@^.@((o.2)"_*regq%#@[)*#@[%_2:
regaic=:+:@(1:+#@(1:,:]))-2:*mll
'program set of regression model'
)
```

左のように、回帰分析に必要な関数群を「大局定義」で行えば、

```
   stat_reg''
program set of regression model
   2 5 $ namelist 3
```

mat	mll	regaic	regb	regcd
regp	regq	regt	resvar	stat_reg

のように「stat_reg」という関数を実行することにより、必要に応じて複数個の関数をまとめて定義することができる。

「namelist 3」は定義されている(代)動詞のセットをボックスの形で出力する。

【観測地点の標高と 07 年 8 月の平均気温】

観測地点	標高(AL：m)	平均気温(TM：℃)
A(甲　府)	273	27.7
B(勝　沼)	394	26.7
C(小　関)	552	24.9
D(河口湖)	860	23.3
E(山　中)	992	21.7
F(富士山頂)	3776	6.4

AL=:273 394 552 860 992 3776 TM=:27.7 26.7 24.9 23.3 21.7 6.4	標高が高くなるほど気温が下がると考えられる。 このデータを用いて、6 合目(2500m)の気温を推定

《１次式のモデル》	《２次式のモデル》	《コメント》
]b1=:TM regb AL 28.5486 0.00592836	]b2=:TM regb AL^/1 2 29.9715 _0.00882574 6.84209e 7	回帰係数が算出されている。
TM regcd AL 99.3547	TM regcd AL^/1 2 99.8732	決定係数の値(%表示)
TM regaic AL 16.3951	TM regaic AL^/1 2 6.63166	情報量規準(AIC)の値で２次式のモデルを選択
+/b1*1 2500 13.7277	+/b2*1,2500^/1 2 12.1835	6 合目での予想気温

年　次	1950	55	60	65	70	75	80	85	90	95
男子自殺者数(百人)	98	138	115	83	88	117	128	154	123	142
完全失業率(%)	1.2	2.5	1.7	1.2	1.1	1.9	2.0	2.6	2.1	3.1

L=:1.2 2.5 1.7 1.2 1.1 1.9 2.0 2.6 2.1 3.1 S=:98 138 115 83 88 117 128 154 123 148		完全失業率と男子自殺者数のデータをそれぞれ(L) (S)に入力している。
《１次式のモデル》	《２次式のモデル》	《コメント》

]b1=:S regb L 52.0889 34.5934	]b2=:S regb L^/1 2 16.1624 74.1546 _9.80831	回帰係数と情報量規準(AIC)を求めている。「ALC」の値は２次式のモデルがやや小さいが、大きいところを予測するのは危険(2次の係数が負)
S regaic L 72.9638	S regaic L^/1 2 69.2763	
+/b1*2 2$1 14 5 190.462 225.056	失業率が４％、５％になった場合の男子自殺者数を予測していて、それぞれ１万９千人、２万３千人弱になるものと予想される。	

```
mean=:3 :'(+/y)%#y' [ sum=:3 :'+/y'
mean_t=:+/%# [ sum_t=:+/
var=:3 :'mean *:dev y.' [ dev=:3 :'y-mean'
var_t=:[:mean[:*:dev=:-mean
sdev=:3 :'%:var y' [ mdev=:3 :'mean|dev y.'
sdev_t=:[:%:var_t [ mdev_t=:[:mean_t[:|dev_t
classfy=:3 :'t,:+/"1(t=.~./:~y)=/y'
meanc=:3 :'(+/*/y)%+/{:y'
varc=:3 :'mean({:y)#*:({.y)-meanc y'
rsk1=:[:+/ sub=:+/\&.|
rsk2=:[:+/[:}.sub^:2
meanr=:4 :'({.x)+({:x)*<:(rsk1%+/)y'
varr=:4 :'(*:x)*((+:rsk2 y)+s-(*:s=.rsk1 y)%n)%n=.+/y'
each=:&>.
div=:3 :'(0=t|y)#t=:1+i.y'
gcd=:4 :'{:/:~(a e. div y)#a=.div x'
lcm=:4 :'{.(b e.a=.x*m)#b=.y*m=.1+i.>.x<.y'
```

NB. 平均と総和(Explicit)
NB. 平均と総和(Tacit)
NB. 偏差と分散(Explicit)
NB. 偏差と分散(Tacit)
NB. 平均偏差と標準誤差
NB. 平均偏差と標準誤差(Tacit)
NB. データを分類する関数
NB. 分類されたデータの平均
NB. 分類されたデータの分散
NB. 第１累積度数の和
NB. 第２累積度数の和
NB. 累積度数法による平均
NB. 累積度数法による分散
NB. イーチ(副詞)
NB. 約数を全て出力(Explicit)
NB. 最大公約数(GCD)
NB. 最小公倍数(LCM)

```
   stat_reg=:3 :0

regb=:[%.1:,.]                                  NB. 回帰係数
regp=:(1:,.])+/ .*regb                          NB. 回帰モデルによる予測値
regq=:[:+/[:*:[-regp                            NB. 残差平方和
regcd=:100"_*1:-regq%[:+/[:*:(]-+/%#)@[         NB. 決定係数(%表示)
mat=:[:%.(|:+/ .*])@(1:,.])                     NB. inverse of data
resvar=:regq%[:-/[:$1:,.]                       matrix
regt=:regb%[:%:resvar*[:(<1 0)&|:mat@]          NB. 残差分散
mll=:>:@^.@((o.2)"_*regq%#@[)*#@[%_2:           NB. 回帰係数のｔ－値
regaic=:+:@(1:+#@(1:,:]))-2:*mll                NB. 最大対数尤度(MLL)
'program set of regression model'               NB. 情報量規準(AIC)
)
```

```
   stat_reg ''
program set of regression model
   2 5 $ namelist 3
```

mat	mll	regaic	regb	regcd
regp	regq	regt	resvar	stat_reg

《主な原始動詞の機能一覧》

演算子	片 側 形	両 側 形
$	右引数のアレイの形を与える	左引数で与えた形に右引数を形成する
#	右引数のアレイのアイテム数を与える	左引数で与えた個数分右引数のコピー
=	自動分類を行う	等しい(論理演算)
=.	変数や関数の局所定義	[機能無し]
=:	変数や関数の大局定義	[機能無し]
>	ボックスを開く	大きい(論理演算)
<	ボックスに入れる	小さい(論理演算)
>.	切り上げて整数値にする	最大値を与える
<.	切り替えて整数値にする	最小値を与える
>:	右引数の数値に１を加える	大きいか等しい(論理演算)
<:	右引数の数値から１を引く	小さいか等しい(論理演算)
+	共役複素数を与える	要素毎の足し算
+.	複素数の実数部と虚数部を与える	最大公約数(論理和)を与える
+:	２倍にする	否定論理和(論理演算のみ)
-	逆符号を与える	要素毎の引き算

-.	論理否定(１を０に、０を１に)	右に含まれないもの(差集合)を与える
-:	２分の１にする	一致していれば１
*	複素平面の単位円周上に射影する	要素毎の掛け算
*.	複素数の極座標を与える	最小公倍数(論理積)を与える
*:	２乗する	否定論理積(論理演算のみ)
%	逆数を与える	要素毎の割り算
%.	逆行列を与える	行列やベクトルの割り算
%:	平方根を与える	右引数の左引数で与えた累乗根
^	指数関数	左引数の右引数で与えた累乗
^.	対数関数	左引数で与えた低の右引数の対数値
^:		重複演算の接続詞
]([]	右引数の内容を表示する	右(左)の要素を取り出す
[:	キャップ	
(	カタログ	左引数の軸のアイテムを出力する
(.	先頭の要素を取り出す	左指定個数分を右引数から取り出す
(:	末尾の要素を取り出す	[機能無し]
}		
).	先頭の要素の落し	左指定個数分を右引数から取り落し
):	末尾の要素の落し	[機能無し]

演算子	片 側 形	両 側 形
,	リスト化する	アイテムを０軸方向に連結する
,.	テーブル化する	高いランクの最高軸方向に連結する
,:	右引数のアレイのランクを１つ増す	高いランクの方の形に合わせ連結する
;	リストのほぐし	ボックスで囲みながら左右を連結
;.		
;:	単語毎にボックスで囲む	[機能無し]
|	(実数や複素数の)絶対値を出力する	整数の割り算の剰余を出力する。
|.	ベクトルや行列の順序を逆にする	ベクトルや行列の回転
|:	軸の総入替え(行列の場合は転置行列)	左で指定した軸に関して転置
/:	昇順のインデクスを与える	昇順に並べ替える

\:	降順のインデクスを与える	降順に並べ替える
“.	数値化する(実行)	右の実行にエラーがあれば左を実行
“:	文字化する	左の複素数で与えた書式で右を表示
~.	重複した要素を排除する	[機能無し]
~:	重複した要素に0を与える	等しくない(論理演算)
!	階乗値を出力する	２項係数を出力する
?	重複を許した整数乱数の生成	重複を許さぬ整数乱数の生成
e.	アトムとオープンの包含積	アトムの包含積
i.	整数列の生成	右引数の要素の左引数の位置を示す
i:	両側整数列の生成	
j.	複素平面上上での 90 度回転	「ajb」は「a+ib」という複素数
o.	円周率「π」	円関数
p.	多項式	
p:	素数	
q:	素因数分解	
r.	右引数を偏角のもつ単位複素数	
x	オイラーの定数	
x:	拡張精度の表示	

《J言語　川柳・都都逸　三十一文字》

・　数値･文字　ボックス表示も　みな「名詞」

・　ともかくも　結果を出さなきゃ　「動詞」じゃない
・　動詞との　出会いひたすら　待つ「副詞」　右にこだわる　「接続詞」
＊　形なき　たった一つは　「アトム」という
＊　横一列　並べアトムよ　これ「リスト」
＊　上から下　リスト集めりゃ　「テーブル」さ
＊　テーブルを　さらに集めて　一般「アレイ」
◇　アトムは0で　リストは1と　各アレイには　「ランク」あり
◇　アレイから　低次のランクの　全てのものを　一括まとめて　「セル」と呼ぶ
◇　１つだけ　ランクの低い　セルだけ特別　「アイテム」といい　動詞が作動
◇　動詞をそのまま　演算すれば　アイテム相手に　作動する
◇　動詞にセルの　ランクをつけりゃ　セルが引数に　早変わり
◇　左右が先で　中の動詞が　後で働く　３連動詞の　「フォーク」なり
☆　二項動詞に　片側動詞が連なれば　引数取り込み　これ「フック」
☆　右から３つで　まずフォーク　左の動詞で　フックを作る（４連動詞）
☆　並んだ動詞は　右からフォーク　残りの動詞と　またフォーク（５連以上の動詞）
・　局所定義は　イコールピリ(=.)　イコールコロン(=:)は　大局定義
・　ボックス(<)で　囲めば全てが　アトムに変身　オープン(>)使って　蘇生する
・　小にピリ(<.)　切り捨てご免で　整数値　大にピリ(>.)なら　切り上げる
・　データから　１を引くなら　小コロン(<:)　１増やすなら　大コロン(>:)
・　不景気で　売上げ半減　マイナスコロン(-:)　プラスコロン(+:)で　所得倍増
・　複素数　実部と虚部は　プラスピリ(+.)　両側形は　最大公約数(GCD)
・　大きさと偏角求める　スターピリ(*.)　両側形は　最小公倍数(LCM)
・　スターコロンは　平方値(*:)　パーセントコロン(%:)は　平方根
・　パーセント(%)　ピリ(.)で一発　行列算　片側形なら　逆行列
・　アレイの形は　ドル($)マーク　アイテム数なら　シャープ(#)さん
・　割算の　余り求める　棒(|)一本　片側形なら　絶対値
・　行列の　逆順・回転　棒にチョン(|.)　コロン(:)付けたら　転置行列
・　ボックスで　囲み連結　セミコロン(;)　片側形なら　リストにほぐす
・　デタラメな　数を生み出す　ハテナキー(?)　重複許さぬ　両側形
・　驚いた(!)　２項係数　瞬時に算出　片側形なら　階乗値
・　だぶってる　データは消せと　ニョロにピリ(~.)
・　ダブルクォート(")　ピリで数値化　コロンで文字化　書式も与える　スグレモノ
・　データを　分類するなら　イコール(=)の　片側形を　使えばよい