（ＪＡＰＬＡ 2007/8/4-6）

「初めてさんのＪ言語(改定版)」

帝京平成大学　鈴木義一郎

【局所定義と大局定義】

局所定義は　イコールピリ(=.)　イコールコロン(=:)で　大局定義

test=:3 :0 a=.i.y. b=:1+a)	test 5 1 2 3 4 5 1+i.5 1 2 3 4 5	右側のボックスで“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左側のボックスで示される。
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。

【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

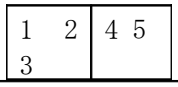
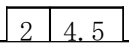
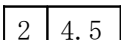
計算を　マトメテ演算　したければ　レベル(L:0)やイーチ(&>)を　使えばよい

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	1x1 2.71828	1x1 ^. 100 4.60517	]ol=:10 ^. 100 2	10 ^ ol 100

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

「4!:0<'def'」は定義内容の品詞を出力する：「0：名詞、1：副詞、2：接続詞、3:動詞」

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

【($ Shape),(# Talley)：アレイの形とアイテム】

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

]A=:2 2	]L=:2 1 2 1	]M=:i.2 3 0 1 2 3 4 5	「行列論」でスカラーに相当するのが「アトム」、横ベクトルが「リスト」で、いわゆる行列が「テーブル」である。	
$ A	$ L 2	$ M 2 3	$ G 2 3 4	]G=:i.2 3 4 0 1 2 3 4 5 6 7 8 9 10 11
# A 1	# L 2	# M 2	# G 2	
+/A 2	+/L 3 +/"1 L 3	+/M 3 5 7 +/"1 M 3 12	+/G 12 14 16 18 20 22 24 26 28 30 32 34 +/"1 G 6 22 38 54 70 86 +/"2 G 12 15 18 21 48 51 54 57	12 13 14 15 16 17 18 19 20 21 22 23

Aのようなスカラーに相当する「アトム」の形は無(次数は0)で、アイテム数は1である。

「]」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」

「+/」はアイテム間に”+”を挿入させることで、「/」は「副詞」である。

A,L,M,Gの次数はそれぞれ0,1,2,3で、そのアイテムの次数はそれぞれ0,0,1,2である。

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	1 2 3	

		4 5 6 (1+i.2 1 3)	3

【# #.#:】

シャープ (#)の　片側形は　タリー(tally)と呼んで　アイテム数を　表示する
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　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

# 3 1	# 3 4 2	# i.3 4 3	「# y.」はアイテム(引数 y.より１つランクの低いセル)数を出力する。
0 1 1 # 1 3 5 3 5	0 2 1 # 1 3 5 3 3 5	]M=:i.2 3 0 1 2 3 4 5	0 1 # M 3 4 5
#.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」である。 「#.」の片側形は２進数を 10 進数に変換	
bi_10 1 0 1 5		bi_10=:3 :'+/y.*2^i.-#y.'	
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	eit_10=:3 :'+/y.*8^i.-#y.'	
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」である。	
]c=:#:3 5 7	#. c	「#:(片側形)」は「#.(片側形)」の逆変換	

0 1 1 1 0 1 1 1 1	3 5 7	

【J 言語の特徴：Fork と Hook】

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)
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)

```
   (sum=:+/)D=:3 1 2
6
   (sum1=:3 :'+/y.')D
6
```

```
   (mean=:+/%#)D
2
   3 :'(+/y.)%#y.'D
2
```

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

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

```
   S % N
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

「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。

```
   (-+/%#)D
_2 _1 0 1 2
```

```
   (dev=:-mean)D
_2 _1 0 1 2
```

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

「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。

```
   ([:(+/%#)[:*:]-+/%#)D
2
```

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

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

```
   ([:(+/%#)[:|]-+/%#)D
1.2
```

```
   ([:mean[:|]-mean)D
1.2
```

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

```
   ([:%:[:mean[:*:dev)D
```

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

	1.41421	1.41421
	７連動詞	５(３)連動詞

【アレイの変形と接続(,,.,:)】

コンマ(,)という　動詞の　片側形は　右引数を　リストに変換

コンマにピリ(,.)の　片側形は　右引数を　テーブル化
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(,)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　両側形は　左右の引数を　横に接続
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

]t=:i.2 3 0 1 2 3 4 5	,t 0 1 2 3 4 5	, i.2 2 2 0 1 2 3 4 5 6 7	「,(Ravel)」という動詞の片側形は、アトムやリストやテーブル等全てが「リスト化」される。
]a=:i.2 0 1	,. a 0 1	,. i.2 2 2 0 1 2 3 4 5 6 7	「,.(Ravel Items)」という動詞の片側形は、右引数の全てのアレイが「テーブル化」される。
,: a 0 1	$,: a 1 2	$,: 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。
「,: a」は見かけは「リスト」のようだが、ランクを１つ上げた「テーブル」である。			

1 2 3 , 4 5 1 2 3 4 5	(A=:i.2 3),b=:6 7 8 0 1 2 3 4 5 6 7 8			「,(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。
1 2 3 ,. b 1 6 2 7 3 8	2,.b 2 6 2 7 2 8	(|:A),.b 0 3 6 1 4 7 2 5 8	(|:A),(,.b) 0 3 6 1 4 7 2 5 8	「,.(Stitch)」という動詞の両側形は「層連結」と呼ばれ、「x,.y」は「x,(,.y)」と同じ結果である。
「,.」による接続は、ランクが増すことも、変わらないこともある				
1,:2	A,:4 5	「,:(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」		

1 2 1 2 3 , : 4 5 6 1 2 3 4 5 6	0 1 2 3 4 5 4 5 0 0 0 0	を付加する。

【不等号と等号含む不等式・切上げと切捨て/ボックスとオープン (< > ; <. >. <: >:)】

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

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

3 < 3.14 1	3 > 3.14 0	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 1	3 <: 3 1	3 = 3.14 0	3 = 3 1	

]B=:<1 2 3 1 2 3	> B 1 2 3	($;#) B | | 1	「ボックス」で囲んだものは「アトム」
]A=:1 2 ; 3 4 1 2 | 3 4	> A 1 2 3 4	; A 1 2 3 4	> 1 2 ; 3 4 5 1 2 0 3 4 5

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

<. 3.14 3	>. 3.14 4	<./ 3 3.14 4 3	>./ 3 3.14 4 4
3 <. 3.14 3	3 >. 3.14 3.14	3([`]@.>)3.14 3	3([`]@.<)3.14 3.14

複数の動詞を「‘（tai）」という接続詞で連結して動名詞(Gerand)を作り、アジェンダ(@.)の右側の関数の値により選択してそれぞれの動詞を作動させている。

「3>3.14」が偽(0)なので「[`]」の０項([]が採択され左側の値を出力する。
「3<3.14」が真(1)なので「[`]」の１項(])が採択され右側の値を出力する。

(+&0.5) 3.14 3.64	(<.@+&0.5) 3.14 3	sgn 3.54 4	sgn=:<.@+&0.5 「四捨五入」関数
<: 3 3.14 2 2.14	>: 3 3.14 4 4.14	(-&1)3 3.14 2 2.14	(+&1)3 3.14 4 4.14

【“+ +. +:”】

実数に　プラス(+)の片側　そのままで　両側形は　フツーの足し算

プラス・ピリ(+.)の　両側形は　最大公約数　複素数でも　イッツオーケー

プラスコロン(+:)　右引数を　倍にする　両側形は　論理演算

+ 0.4 0 3 0.4 0 3	+ 3j4 3j_4	1 2 3 + 4 5 6 5 7 9	複素数に対して「+」の片側形は共役複素数
cdiv=:4 :'{:/:~(a e. b=.div y.)#a=.div x.'		「最小公倍数(ＬＣＤ)」を出力する関数を定義している。	
6 +. 8 2	6 cdiv 8 2	6 +. 5 1	6 cdiv 5 1
]b=:+: a=:1 2 3 2 4 6	2 * a 2 4 6	-: b 1 2 3	「-:」は「+:」の逆演算

0 +: 0 1	0 +: 1 0	1 +: 0 0	1 +: 1 0	0 0 1 1([:-.*@+)0 1 0 1 1 0 0 0
0 0 1 1(1:`0:@.*@+)0 1 0 1 1 0 0 0				左右の引数が共に「0」のときだけ「1」を出力する。
0 *. 0 0	0 *. 1 0	1 *. 0 0	1 *. 1 1	0 0 1 1(*@*)0 1 0 1 0 0 0 1
0 0 1 1(0:`1:@.*@*)0 1 0 1 0 0 0 1				左右の引数が共に「1」のときだけ「1」を出力する。

【- -. -:】

マイナス(-)の 片側形は 符号の反転　両側フツーの　引き算よ

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　右引数に　無いモノを出す
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　「形」まで含めて　一致(1)か否(0)か

- _5 0 2 5 0 _2	4 5 6 - 1 3 2 3 2 4	- 3j4 _3j_4	3j4 - 1j1 2j3
]b=:-.a=:4 0.3 1 _3 0.7 0	a + b 1 1 1	]d=:-. c=:3j4 _2j_4	c + d 1
a=:1 2 b=:1 3 5	a -. b 2	]c=:-.a e.b 0 1	c # a 2
b -. a 3 5	(-.b e.a)#b 3 5	左の引数から右引数の要素を除去したものを出力する。	
(a e. b)#a 1 2	{:/:~(a e. b)#a 2		
]a=: -: 2 4 6 1 2 3	2 4 6 % 2 1 2 3	(halve=:%2:)2 4 6 1 2 3	+: a 2 4 6

1 2 -: 1 2 1	1 2 = 1 2 1 1	1 2 -: 2 2 0	1 2 = 2 2 0 1
match=:[:*/=	1 2 match 1 2 1	1 2 match 2 2 0	1 2 -: 1 2 1 2 0
1 2 ~: 1 2 0 0	1 2 ~: 2 2 1 0	1 2 not_equal 1 2 0 0	1 2 not_equal 2 2 1 0
not_equal=:[:-.=		等しいときに「0」等しくないときに「1」を出力する。「~:」は ‘not equal’	

【＊ ＊. ＊:】

シグナム(＊)は　符号(_1 0 1)与える　片側形　両側フツーの　掛け算よ
スター・ピリ(＊.)の　両側形は　最小公倍数
スター・コロン(＊:)は　平方値　平方根なら　パーセント・コロン(%:)
スター・コロン(＊:)の　両側形は　論理演算　１と１なら　０を出力

＊ _3 0 2 _1 0 1	_3 2 ＊ 2 _6 4	_3 2 ＊ 2 4 _6 8	
4 ＊. 6 12 _4 ＊. 6 _12 _4 ＊. _6 12	4 lcm 6 12 _4 lcm 6 _12 _4 lcm _6 12	lcm=:4 :0 a=. p*m=. 1+i. >. (p=. |x.)<. q=. |y. (*x. *y.)*(b e. a)#b=. q*m)	
＊: 4 16	%: 16 4	＊:b. _1 %:	%:b. _1 ＊:

L=:0 0 1 1 [R=:0 1 0 1			
L *. R 0 0 0 1	L (1:<+) R 0 0 0 1	L 1:`0:@. (2&>@+) R 0 0 0 1	左右の引数が共に「1」のときだけ「1」を出力する。
L *: R 1 1 1 0	L (1:>:+) R 1 1 1 0	L (1:`0:@. *) R 1 1 1 0	左右の引数が共に「1」のときだけ「0」を出力する。
L +. R 0 1 1 1	L (0:<+) R 0 1 1 1	L 0:`1:@. (0&<@+) R 0 1 1 1	左右の引数が共に「0」のときだけ「0」を出力する。
L +: R 1 0 0 0	L (0:>:+) R 1 0 0 0	L (1:`0:@. *@+) R 1 0 0 0	左右の引数が共に「0」のときだけ「1」を出力する。

【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

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	の形で生成している。「?.」を使えばシードを固定できる。

【% %. %:】

パーセント(%)　片側形は　逆数で　両側形なら　フツーの割り算

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

% 2 4 5 0.5 0.25 0.2	2 4 5 % 2 1 2 2.5	4 9 % 2 3 2 3	「%」の片側形は逆数で、両側形は割算。
14 40 %. A=:2 2$1 1 2 4 8 6		「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？」	
]B=:%. A 2 _0.5 _1 0.5	A +/ .* B 1 0 0 1	A +/ .* 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
%: 2 4 9 1.41421 2 3	3 %: 8 27 2 3	0.5 %: 2 3 4 9	*: 2 3 4 9

【副詞は後から　役割果す　動詞の活躍　拡大す】

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)

+/~ a=:1+i.3 2 3 4 3 4 5 4 5 6	a +/ a 2 3 4 3 4 5 4 5 6	*/~ a 1 2 3 2 4 6 3 6 9	a */ a 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
a =: 1 2 1 3 3 2 1	~. a 1 2 3	~:a 1 1 0 1 0 0 0	(~:a) # a 1 2 3	(~.nub)の片側形は重複要素の排除
(1 2)~:2 1 1 1	(1 2)~:1 2 0 0	(1 2)~:2 1 3 length error	(~:)の両側形は各要素毎の不一致に“1”
(1 2)-:2 1 0	(1 2)-:1 2 1	(1 2)-:2 1 3 0	(-:)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

2 3 [4 5 2 3	2 3] 4 5 4 5		
([:*:+:)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 4	2(+*-)1 3		
(abs=:| :[:)_1 2 3 1 2 3	2(res=:[: :|)1 2 3 1 0 1	キャップ([: cap])は　演算結果に関係せず	

【 { } {. {: 】

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

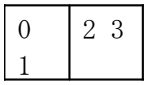
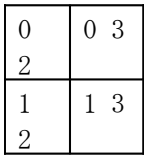
中カッコ({}　左で与えた　インデクスの　アイテムを取る　両側関数

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)

ヘッド({.}で先頭　テール({:)で末尾　要素取り出す　片側動詞

左で与えた個数分　take({.}は取りで　drop(}.)は除く　便利な両側　動詞なり

<table>
<tr><td>0 1 ; 2 3
  </td><td>{3
3 アトムに対しては“ボックス(<)”と同じ</td><td>{ 0 1 ; 2 3
  </td><td></td></tr>
<tr><td>1 { 1 2 3
2</td><td>0 { i.2 3
0 1 2</td><td>]M=:i.2 3
0 1 2
3 4 5</td><td>1 1 0 } M
3 4 2</td></tr>
<tr><td>3 4(0 1)}i.3
3 4 2</td><td>'BD'(1 3)}A=:'abcde'
aBcDe</td><td>A
abcde</td><td>修正したものを定義し直さないと変更されない。</td></tr>
<tr><td>]K=:1+i.3
1 2 3</td><td>{. K
1</td><td>{: K
3</td><td>_1 {. K
3</td></tr>
<tr><td>2 {. K
1 2</td><td>2 }. K
3</td><td colspan="2">「}:」の両側形は機能無し。</td></tr>
</table>

【“i. i:”】

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.3 0 1 2	i._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。
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)。
icolon=:3 :'|.^:(-:1+*y.)(|.,-@}.)i.1+|y.'		“i:” を使わず同じ働きの関数。

]a=.i.1+| 3 0 1 2 3	]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	(* 3) 1	(*_3) _1	(-:1+*3) 1	(-:1+*_3) 0
|.^:1(|.,-@}.)a _3 _2 _1 0 1 2 3	|.^:0(|.,-@}.)a 3 2 1 0 _1 _2 _3	「|」：絶対値をとる 「}.」：先頭の要素を落とす 「-」：符号の逆転「-:」：数値を半分に 「|.」：全要素(アイテム)の反転	
icoron 3 _3 _2 _1 0 1 2 3	icoron _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。			

'abcdef' i. ' aec' 6 0 4 2			(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(空欄には６)
'abcae' i. 'aceg' 0 2 4 5			左の要素にない” g”にはインデックス外の数値(5)が表示される。
$ a. 256	a. i. 'Aa' 65 97	65 97 { a. Aa	「a.」には256個の文字やキャラクターが入力されている。
'abcae' i: 'aceg' 3 2 4 5	a. i: 'A a' 65 32 97		(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

【指数関数と対数関数/オイラーの定数(^ ^. ^: x)】

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　左の数だけ　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

^ i.3 1 2.71828 7.38906	1x0 1x1 1x2 1 2.71828 7.38906	*:123456789 1.52416e16	*:123456789x 152415787501905 21
(1x1)^ i.3 1 2.71828 7.38906	2 3 ^ 3 4 8 81	^&2(2 3) 4 9	*: 2 3 4 9
^. 1 2 0 0.693147	(1x1)^. 1 2 0 0.693147	5 10 ^. 125 100 3 2	5 10 ^ 3 2 125 100
>:^:2 i.3 2 3 4	>:>: i.3 2 3 4	>:^:_1 (2 3 4) 1 2 3	<: 2 3 4 1 2 3

【マイナス、無限大と不定形(_ _. _:)】

- 3.14 _3.14	- _3.14 3.14		アンダー・バー(_)は　負数を示す　名詞 無限大(∞)も示す　両刀使い
2 % _ 0	2 % __ 0	_ + _ _	
_ - _ _.	3 + _. _.	2 * _. 0	“_.”は　不定形(indeterminate)の名詞
_: '' _	(]%_:)2 0		「_:」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。

【(| |. |:)】

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

3 | i.6 0 1 2 0 1 2	| 1 _2 3 _4 1 2 3 4	| 3j4 5	
|.d=:1 2 3 4 5 5 4 3 2 1	|. i.2 3 3 4 5 0 1 2	「|.」の片側形はアイテムの順序の逆転	
1 |.d 2 3 4 5 1	_1 |. d 5 1 2 3 4	2 |.d 3 4 5 1 2	_2 |. d 4 5 1 2 3
|:i.3 3 0 3 6 1 4 7 2 5 8	]m=:2 3$'abcdef' abc def	|:m ad be cf	1 0|:m ad be cf
1 |: m abc def	0 1|:m abc def	(<0 1)|:m ae	(<0 1)|: i.3 3 0 4 8

【I.(Indices)】

]b=:i.@# a=:0 0 1 0 1 0 0 1 2 3 4 5	a # b 2 4	(indices=:#i.@#)a 2 4	I. a 2 4
	(-.a) # b 0 1 3 5	indices -.a 0 1 3 5	I. -.a 0 1 3 5
]d=:(i.@#)c=:0 0 1 0 2 0 0 1 2 3 4 5	c # d 2 4 4	indices c 2 4 4	I. c 2 4 4

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

【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

Even(..)やOdd(.:)の　接続詞　２つの動詞を接続　別の動詞を　作り出す

*:&+: 2 3 4 16 36 64	*:@+: 2 3 4 16 36 64	*:@:+: 2 3 4 16 36 64	
*:&.+: 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「u の逆演算」が実行される。	
2&* 2 3 4 4 6 8	(*&2)2 3 4 4 6 8	*: 2 3 4 4 9 16	
+`*/ i.6 29	0+1*2+3*4+5 29	+`%/ 3 1 4 3.25	3 + 1 % 4 3.25
(+:`-:`:0)3 2 6 6 4 12 1.5 1 3	(+:,:-:)3 2 6 6 4 12 1.5 1 3	(u`v`:0)は　全ての動詞を　演算する。	
(+`*`: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
(+`*`-`: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	

【: :. ::】

log=:10&ˆ. : ˆ. log 10 100 1 2	8 log 10 100 1.10731 2.21462	単項と 2項の動詞を コロン(:)で結べば 同時に定義 できますよ
f=:*: :. %: f i.5 0 1 4 9 16 fˆ:_1 f i.5 0 1 2 3 4	g=:*: :. +: g i.5 0 1 4 9 16 gˆ:_1 g i.5 0 2 8 18 32	Obverse(:.)は 逆が正しい 定義なら 逆変換(ˆ:_1)で 元に返る(関数“f”) (“g”の定義関数では“:.”の右の関数が 左の関数の逆関数でないので右の関数の演算結果が表示される)
p=:3 1 0 2 q=:3 1 1 0 test=:A. ::(!@#) test p 20	test q 24 A. q index error	「u :: v」は エラーが無ければ “u” エラーがあれば “v”の演算結果を出力 (“::”は「Adverse」という接続詞)

【″ ″. ″:】

X=:7 8 9 [Y=:i.2 3 X,″1 Y 7 8 9 0 1 2 7 8 9 3 4 5	X,″2 Y 7 8 9 0 1 2 3 4 5	+/i.2 3 3 5 7 +/″1 i.2 3 3 12	ダブルクォート(″)はランク指定の接続詞
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)	4.2 ″: 3.14159 3.14 5.3 ″: 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	8 ″. b 1 2 3 4 5 8 8 8 8	「″.」の両側形は隙間に左の数値を挿入	

3 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	(1.5*2.5)*-:%:1p1 3.32335

2 ! 5 10	(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

f0=:^!.0 3 f0 2 9	3^2 9	2 f0 3 8	2^3 8	f0~ 2 3 4 27 256	^~ 2 3 4 4 27 256

（どんな目的で、このような関数が必要なのかは不可解！）

g0=:[:*/[+]#0: h0=:[:*/[+0:*i.@]	g0"0~c=:2 3 4 4 27 256	h0"0~c 4 27 256	

f1=:^!.1 f2=:^!.2	g1=:[:*/[+]#1: g2=:[:*/[+]#2:	h1=:[:*/[+1:*i.@] h2=:[:*/[+2:*i.@]	

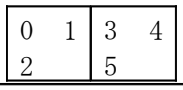
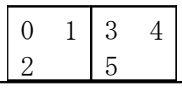
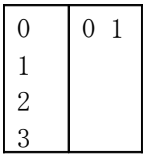
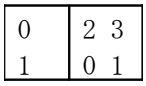
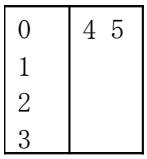
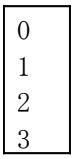
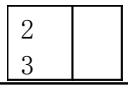
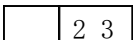
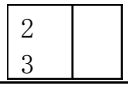
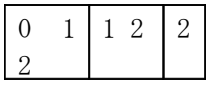
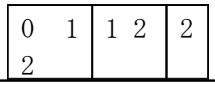
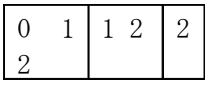
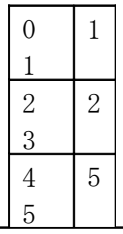
f1~ c 6 60 840	g1"0~c 6 60 840	h1"0~c 6 60 840
f2~ c 8 105 1920	g2"0~c 8 105 1920	h2"0~c 8 105 1920

f_1=:^!._1 f_2=:^!._2	g_1=:[:*/[+]#_1: g_2=:[:*/[+]#_2:	h_1=:[:*/[+_1:*i.@] h_2=:[:*/[+_2:*i.@]	

f_1~ c 2 6 24	g_1"0~c 2 6 24	h_1"0~c 2 6 24
f_2~ c 0 _3 0	g_2"0~c 0 _3 0	g_2"0~c 0 _3 0

“ 0!:n”, “1!:n”,…… , “ 7!:n”, “9!:n”, “11!:n”, “13!:n”, “14!:n”, “15!:n”

“128!:0”, “128!:1”は、外部接続詞で、いろいるなシステム関数が用意されている。

; i.2 3 **0 1 2 3 4 5**	;/ i.2 3 	0 1 2 ; 3 4 5 	セミコロン(;)の片側形は　右のアレイをリストにほぐす
];.0 i.2 2 3 2 1 0	|.″1|. i.2 2 3 2 1 0	「];.0」は　全ての軸を　逆順にする。	
2 2];.0 i.3 2 0 1 2 3	2 2{. i.3 2 0 1 2 3	2 1];.0 i.3 2 0 2	2 1{. i.3 2 0 2
]M=:3 2 $ i.4 0 1 2 3 0 1	<;.1 M 	<;.2 M 	「<;.1」は先頭で「<;.2」は末尾で、フレット(0 1)の表れた位置で切る
1 0 1 <;.1 i.3 2 	0 1 0 <;.2 i.3 2 	「<;.1」や「<;.2」の両側形は先頭や末尾から1が現れると　区切る	
<;._1 (3 2$i.4) 	<;._2 (3 2$i.4) 	「<;._1」は先頭、「<;._2」は末尾からフレットを除いて　区切りを入れる	
1 0 1 <;._1 i.3 2 	0 1 0 <;._2 i.3 2 	「<;._1」や「<;._2」の両側形は「<;.1」や「<;.2」の先頭や末尾を削除する	
		「;.3」片側形は	
<;.3 i.3  <¥&.|.i.3 	3 <;.3 i.3  <;.3 A	2 2 <;.3 i.3 2	1 2 <;.3 i.3 2 

]A=:2 2$’ abcd’
ab
cd

ab cd	b d
cd	D

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

データに　プラス・スラッシュ(+/) 合計算	+/ 3 1 2 6
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)	</.i.3 4 0 | 1 4 | 2 5 8 | 3 6 9 | 7 10 | 11
イコール(=)記号の　片側形は　分類・集計に　チョー便利	a=:1 2 3 1 3 2 1

]b=:= a 1 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0	(~.a)=/a 1 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0	+/"1 b 3 2 2	(~.a),:+/"1 b 1 2 3 3 2 2
b <@# A=:'abcdefg' adg | bf | ce		a </. A adg | bf | ce	
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		/:~ 3 1 2 1 2 3	¥:~ 3 1 2 3 2 1

*: .. +: 4 40	-:@(*:+*:&+:) 4 40	*: .: +: 4 _24	-:@(*:-*:&+:) 4 _24
+: .. *: 4 20	-:@(+:++:&*:) 4 20	+: .: *: 4 12	-:@(+:-+:&*:) 4 12
>: .. <: 5 5.5	-:@(>:+>:&<:)5 5.5	>: .: <: 5 0.5	-:@(>:->:&<:)5 0.5
<: .. >: 5 4.5	-:@(<:+<:&>:)5 4.5	<: .: >: 5 0.5	-:@(<:-<:&>:)5 0.5

(9!:3)2 4 5 mean + | / | % | # +- / --- + --+- %		4!:0 'a';'mean' _1 3 (4!:1)3 +----+ |mean| +----+	

+- # +/ % #			

【$. (Sparce)】

+:&.$. M=:i.2 3 0 2 4 6 8 10	*:&.$. M 0 1 4 9 16 25	(+ : M) -: +:&. $. M 1 (* : M) -: *:&. $. M 1	
0 $. M 0 1 | 1 0 2 | 2 1 0 | 3 1 1 | 4 1 2 | 5	+&.($.^:_1)M 0 1 | 1 0 2 | 2 1 0 | 3 1 1 | 4 1 2 | 5	(+:1+M) -: +:&. $. 1+M 1	
0 $. 1+M 0 0 | 1 0 1 | 2 0 2 | 3 1 0 | 4 1 1 | 5 1 2 | 6	+&.($.^:_1)1+M 0 0 | 1 0 1 | 2 0 2 | 3 1 0 | 4 1 1 | 5 1 2 | 6		
+:&.($.^:_1)i.5 1 | 2 2 | 4 3 | 6 4 | 8	0$.i.3 1 | 1 2 | 2	0$.1+i.3 0 | 1 1 | 2 2 | 3	

【$:(Self Reference)】

1:`(]*$:@<:)@.* 5 120	5*4*3*2*1 120	フレーズの生じた結果を代理して受け、左の連結詞や文の実行が完了したときに停止する。「5×4×3×2×1」や「5+4+3+2+1」
0:`(]+$:@<:)@.* 5 15	5+4+3+2+1+0 15	
0:`(]-$:@<:)@.* 5	5-4-3-2-1-0	

3	3	