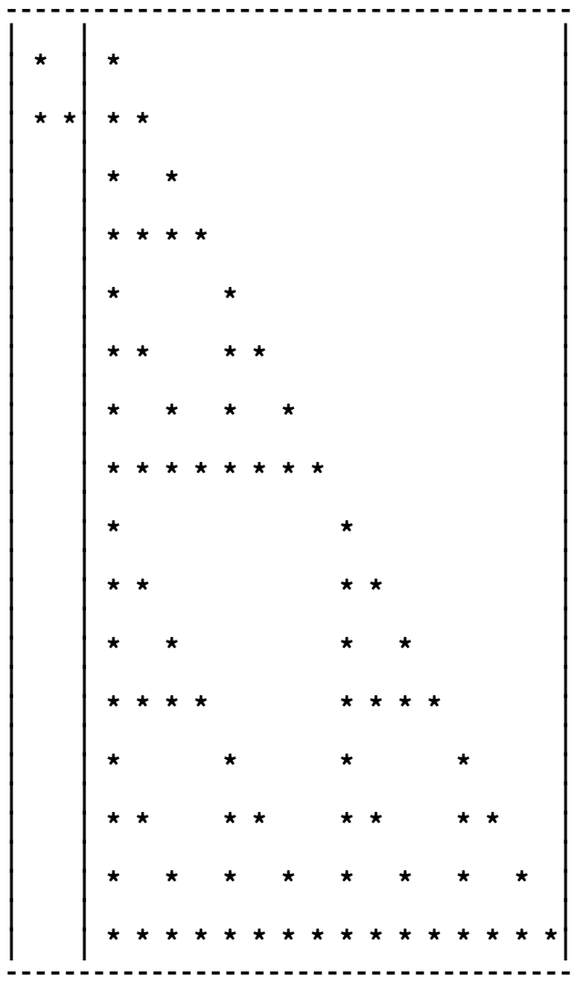
Ｊ言語のためのｸｲｯｸ･ﾚﾌｧﾚﾝｽ
《QUICK REFERENCE FOR J-LANGUAGE》

```
NB.Sierpinsky's gasket
unit=. (' '(<2 1)}]\4$' *')"_
gasket=. (],],.]) ^:[unit
(unit'');gasket 3
```



【日本ＡＰＬ協会】

Ｊ言語研究会編

はじめに …… 平成時代の新言語

人と人とが意志を伝えあうためには、まず言葉が必要である。コンピュータに仕事をさせる場合でも、何らかの方法でコンピュータとのコミュニケーションを図る必要がある。コンピュータが理解できる言葉は、端的にいえば、０と１の羅列からなる「機械語」である。この機械語を人間がみると、まるで暗号のようなもので、これではコンピュータと会話することはまず不可能である。そこで、人間が理解しやすい言語がいろいろと考案されてきた。最もポピュラーな言語としては『ＢＡＳＩＣ』があるが、昨今ではより高水準の『Ｃ』や『Ｐａｓｃａｌ』といった言語のほうが多用されつつある。特に科学技術計算向きの言語として、４０年ほど前に開発されたのが『ＡＰＬ』という言語であるが、当時としては大型計算機でないと利用することができないことや、見馴れない特殊記号を用いていたために、広く普及するには至らなかった。

ところが１０年ほど前に、ＡＰＬの創始者であるアイバーソン(K.E.Iverson)を中心として、「ＡＰＬ」の泣きどころであった特殊記号を排し、通常のアスキー記号にピリオド「．」やコロン「：」を付加することによって、記号の種類を３倍に拡大し、ＡＰＬの機能をさらに強力にした「関数型言語」を開発し、これを言語『Ｊ』と名づけた。コンピュータの飛躍的な進歩は、パソコンレベルでもＡＰＬやＪが使える時代になったのである。ともあれ「Ｊ」言語は、パソコンを「超」高級電卓として利用できることが大きなメリットといえる。つまり、コンピュータに不慣れなタイプでも、すぐに自分でプログラムを作ることができる。そこで、この「Ｊ」言語なるものの簡単な紹介を試みてみたい。とにかく、「馬には乗ってみよ、人には添うてみよ」というように、是非、チャレンジしてみて頂きたい。

本稿は、Ｊ言語の記号と基本概念を、機能・用途別に分類し、簡単な説明を加えた『クイック・リファレンス』である。Ｊ言語の特徴と概要を知る上で便利なものであり、またＡＰＬを知らない読者でも初めてＪ言語を学ぶときにも役立つものと思う。

Ｊ言語のキーワードの日本語表現については、Ｊ言語研究会で議論・統合したものを用いた。対応するＡＰＬの説明や用例は、『日本語ＡＰＬクイック・リファレンス(日本ＩＢＭ社)』と対比し、参照した。

本書の内容は、リリース４(1994.4.1)に準拠している。リリース２では、アメンド(}　)やコントロールシークエンス(Ｓ．)などがリリース１から大幅に変更された。リリース３では、分数や長い実数の表現が加えられ、また数値計算の拡張や精度の向上が図られ、更に関数の追加、改良も加えられた。リリース３からリリース４への変更では、単項のランクの多くが無限大から０に変更された他は、言語機能や文法についてはほとんど変更は認められない。

《目次》

《QUICK REFERENCE FOR J-LANGUAGE》

EDITTED BY
J RESEARCH GROUP (JAPAN APL ASSOCIATION)

《CONTENTS》

第０章　Ｊ言語の特徴

品詞

　Ｊ言語では、ＡＰＬでの変数・関数等の用語に対して、以下のような自然言語における品詞の名前を用いている。【「代名詞」や「代動詞」という概念もあるが、特に名詞、動詞と区別しなくともよい】

（Ｊ言語）	（ＡＰＬ）
名詞	定数
動詞	関数
副詞	作用素
接続詞	ー

数学用語との比較

Ｊと数学用語との対応表を以下に示す。Ｊは数値計算以外にも事務計算や文字列の処理などにも利用できる汎用言語であるから、必ずしも数学用語とは一致しない。

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

名詞

型の指定は必要ない。種類としては　数値、　文字、　ボックス

数値はそのまま定義する。

```
A=.1 3 5
even=.2 4 6
```

文字列はクオート('')で囲んで定義する。

```
SEI=.'SHIMURA'
```

数値や文字を四角で囲んだものがボックス(Box)として定義できる。ボックスと数値は直接は演算できない。ただボックスの中で演算させることはできる。

```
< A                < SEI
```

1 3 5	SHIMURA

配列の大きさは「$」で指定する。

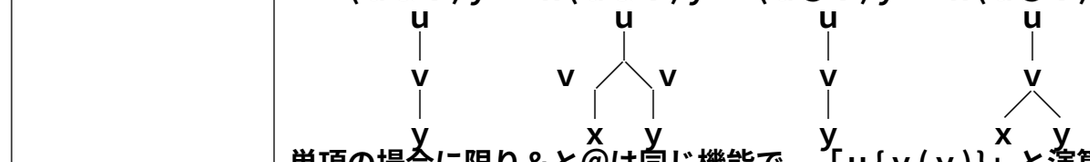
```
   2 3 $ 5 2 3 4 1 6
5 2 3
4 1 6
```

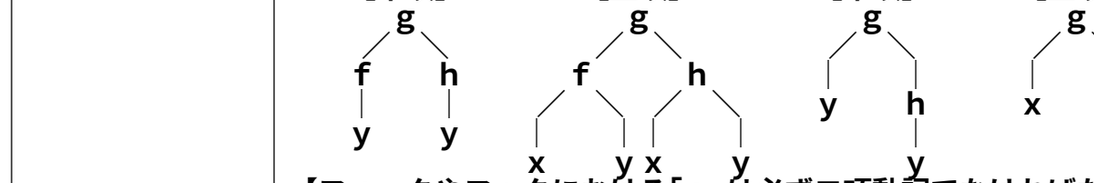
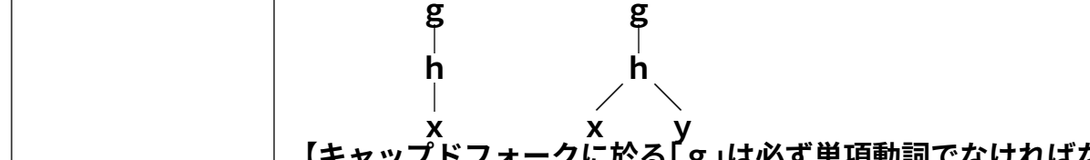
０から始まる整数列の生成には「i.」を用いればよい。

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

インクルードファイルの指定	ファイルの指定は特に必要としないが、“profile.ijs”を読み込むとき必要とする関数を指定するほうがよい。 (profile.ijs のサンプル) host=.0!:0 in=.0!:2@< over=.({.,.@;}.)@":@, by=.''&;@,.@[,.] in 'c:＼jpc＼js＼mvar.ijs'
プリミティブ	Ｊ言語で用いられる、＋や－：等の記号を「プリミティブ」といい、動詞、副詞、接続詞などがある。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ってもよい。このとき、‘plus’や‘squre’は「代動詞」と呼ばれる。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。
単項と二項	ほとんどの動詞には、(右引数だけの)単項と、(左右に引数をとる)二項の２種類がある。キーボードに制約があるため、単項と二項とを兼用する場合があり、使い分けに注意する必要がある。 ^. 100 4.60517　（自然対数） 10 ^. 100 2　　（常用対数）
数値	数値は全て明記する必要があり、また“.5”のように“０”を省略することができず“0.5"のように入力しなければならない。さらにリストやテーブルを定義するときも、数値の間にはスペースを入れなければならない。負の数にはアンダーバーをつけて“_5”のように表わす。ＡＰＬの場合のオーバーバーに対応するもので“¯”とは異なる点に注意。
スペース	プリミティブと数値の間にはスペースをとる必要がないが、代動詞と数値の間には必ずスペースが必要である。ドットやコロンが独立して使われる場合には、前のプリミティブとの間にスペースが必要となる。また明示的定義を宣言する際の「name=.3 :0」では、コロンの前のスペースは絶対必要。さらに 内積の「+/ .*」の場合もドットの前のスペースは必ず挿入しなければならない。
プリミティブの優先順位	全てのプリミティブの優先順位は平等である。 代動詞(ＡＰＬでは定義関数)の定義には　明示型と　タシット型がある 明示型定義での演算の順序は、右から順に演算する。 5%4*3+2-1 0.3125　　【「5%(4*(3+(2-1))) = 0.3125」のように演算する】 タシット型での合成関数の定義では、フォーク(Fork)やフック(Hook)を作るので、その順序関係はやや複雑になる。 5(+%-)3 4　　【「(5+3)%(5-3) = 4」のように演算する】

副詞	副詞は(右から)動詞を修飾して新たな動詞を作る。
	最も縦横に活躍する副詞は「スラッシュ(/)」である。
	`   +/ 1 2 3 4 5`
	`15`　　【「1+2+3+4+5 = 15」のように演算する】
	`   */ 1 2 3 4 5`
	`120`　　【「1*2*3*4*5 = 120」のように演算する】

接続詞	Ｊでは、動詞と動詞、あるいは動詞と名詞を連結して新たな動詞を作ることができる。その連結用に「接続詞」というものを導入した。その代表が、次の「ボンド(&)」と「アトップ(@)」である。 《＆》　　《@》 【単項】　【二項】　【単項】　【二項】 (ｕ＆ｖ)ｙ　ｘ(ｕ＆ｖ)ｙ　(ｕ@ｖ)ｙ　ｘ(ｕ@ｖ)ｙ  単項の場合に限り＆と@は同じ機能で、「ｕ{ｖ(ｙ)}」と演算する 二項の場合は「ｘ(ｕ＆ｖ)ｙ」＝「ｖ(ｘ)ｕｖ(ｙ)」 「ｘ(ｕ@ｖ)ｙ」＝「ｕ(ｘｖｙ)」 動詞と名詞の結合には「ｕ＆m」，「ｎ＆ｖ」のように＆を用いる。

フォークとフック	タシットによる定義で動詞を関数合成により連結する場合には、３連動詞(フォーク)が最優先される。２つの場合にはフックになる。
	`   (+/%#)a=.1 2 3 4 5`
	`3`　　【「(+/ a)%(#a) = 3」のように演算する】
	`   5(+%-)3`
	`4`　　【「(5+3)%(5-3) = 4」のように演算する】
	`   (*-)3`
	`_9`　　(「3*(-3) = _9」のように演算する)
	《フォーク(ｆｇｈ)》　　《フック(ｇｈ)》
	【単項】　【二項】　【単項】　【二項】
	 
	【フォークやフックにおける「ｇ」は必ず二項動詞でなければならない】
	《キャップドフォーク(［：ｇｈ)》
	【単項】　【二項】
	 
	【キャップドフォークに於る「ｇ」は必ず単項動詞でなければならない】

第１章　数値計算の概要

１.１　基本的な数学演算

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
+ Conjugate 共役 + Y	＋	実数の範囲内ではそのまま 　+ 0.4 _5 0 0.4 _5 0 　+ 3j4 3j_4(共役複素数)	Plus 加算 X + Y	左引数と右引数との和 　5 + 1 2 3 6 7 8 　1 2 3 + 4 5 6 5 7 9
- Negate 逆符号	－	右引数の符号を反対にする 　- _5 0 1 5 0 _1 　- 3j4 _3j_4 実部と虚部の符号を逆にする	Minus 減算 X - Y	左引数から右引数を引く 　8 _3 - 6 _5 2 2 　3.5j0.5 - 1j1 2.5j_0.5 複素数どうしの引算も可能
* Signum 符号 * Y	×	実数に対しては、正には１、０には０、負には－１を付与 　* _5 0 4 _1 0 1 　* 3j4 0.6j0.8 　*. 3j4 5 0.927295 (大きさと偏角を与えている) 　*. 0.6j0.8 1 0.927295 （複素数に対しては、大きさが１で、同じ偏角の複素数）	Times 乗算 X * Y	左引数と右引数との積 　0 1 * 3 4 0 4 　3.5j0.5 * 1j1 3j4 複素数どうしの掛算も可能
% Reciprocal 逆数 % Y	÷	「% y」は「1 % y」と同じ。 　% 4 0.25 　% 3j4 0.12j_0.16 　3j4 * % 3j4 1 　*. 3j4 5 0.927295 　*. % 3j4 0.2 _0.927295 複素数の逆数は大きさが逆数で、偏角が反対符号になる	Devide-by 除算 X % Y	左引数を右引数で割る 　0 8 % 1 0.4 0 20 　3j4 % 1j1 3.5j0.5 　1j1 * 3j4 % 1j1 3j4 複素数どうしの割算も可能
Magnitude 絶対値 　Y		符号をとって絶対値にする 　　6 _5 6 5 　　3j4 5 複素数に対しては大きさ	Residue 剰余 X　Y	右引数を左引数で割ったときの余り 　3　i.7 0 1 2 0 1 2 0 　3　-i.7 0 2 1 0 2 1 0 　_3　i.7

				0 _2 _1 0 _2 _1 0 _3 -i.7 0 _1 _2 0 _1 _2 0
+: Double ２倍 +: Y		右引数の要素を２倍にする +: i.5 0 2 4 6 8 2 * i.5 0 2 4 6 8	Not-Or 否定論理和 X +: Y	左右の引数が共に０のときに１、そうでなければ０ 0 0 1 1 +: 0 1 0 1 1 0 0 0
-: Halve ２分の１ -: Y		右引数の要素を半分にする -: 0 2 4 0 1 2 0 2 4 % 2 0 1 2	Match 一致 (2.1)	
*: Square 平方 *: Y		右引数の要素を平方する *: i.5 0 1 4 9 16	Not-And 否定論理積 X *: Y	左右の引数が共に１のときに０、そうでなければ１ 0 0 1 1 *: 0 1 0 1 1 1 1 0
%: Square root 平方根 %: Y		右引数の要素の平方根をとる %: 4 9 16 2 3 4 %: _4 0j2 負の値のときは虚根になる	Root 累乗根 X %: Y	ＹのＸ乗根で「Y ^ % X」に等しい 2 3 4 %: 4 27 256 2 3 4 _1 %: 4 0.25 －１乗根は逆数に等しい
^ Exponent -ial　指数 ^ Y	＊	ｅのＹ乗 ^ 0 1 2 1 2.71828 7.38906	Power 累乗 X ^ Y	ＸをＹ乗するに等しい 2 3 4 ^ 1 2 3 2 9 64
^. Natural -Log 自然対数 ^. Y		底がｅのＹの対数(自然対数) ^. 1 2 1 0.693147	Log 対数 X ^. Y	Ｘを底とするＹの対数に等しい 10 ^. 125 100 3 2

>: Incremment １増 >: Y	≧	>: i.5 1 2 3 4 5 (i.5)+1 1 2 3 4 5	Larger or equal 大きいか等しい X >: Y	左引数が右引数より大きいか等しければ１、そうでなければ０ 20 30 40 >: 40 30 20 0 1 1
<: Decrement １減 <: Y	≦	<: i.5 _1 0 1 2 3 (i.5)-1 _1 0 1 2 3	Lesser or equal 小さいか等しい X <: Y	左引数が右引数より小さいか等しければ１、そうでなければ０ 20 30 40 <: 40 30 20 1 1 0
>. Ceiling		Ｙより大きな最小の整数 >. 0.4 _0.3 _3.4	Larger of 最大値	左引数と右引数の大きい方の値を選ぶ

天井値 >. Y		1 0 _3 (切上げ) >./ 1 2 3 4 5 5 (最大値の取り出し)		4 0.5 >. 1 6 4 6
<. Floor 床値 <. Y		Yを超えない最大の整数 <. 0.4 _0.3 _3.4 0 _1 _4 (切捨て) <./ 1 2 3 4 5 1 (最小値の取り出し)	Lesser of 最小値 X <. Y	左引数と右引数の小さい方の値を選ぶ 4 0.5 <. 1 6 1 0.5

１.２　マイナス、無限大

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
_ Negative -sogn マイナス Infinity 無限大		数学のマイナス記号(形容詞) - 3.14 _3.14 単独では無限大を表わす名詞 2 % 0 _	（無し）	
_. Indetermi- nate 不定		無限大の演算で生じる不定形を表わす名詞 _ - _ _. (3+_),3+_. _ _.	（無し）	

_: Infinity 無限大		無限大の定数を生じさせる動詞 _: 1 2 3 _	Infinity 無限大	右引数の指定したランクに対応した形の無限大を派生 (_:"0)i.2 3 _ _ _ _ _(_:"1)i.2 3 _ _3.14(_:"1)i.2 3 左引数には関係ない

１.３　円関数

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
o. Pi-times 円周倍率 o.Y		πとYとの積（Yπ） (o.1),o.2 3.14159 6.28319	Circular Function 円関数 X o.Y	左引数で円関数の型を指定 rdf=.%&180@o. rdf 90 180 1.5708 3.14159 (角度をラジアンに変換) 1 o. 1.5708 1 (SIN 90°の値) 2 o. 1.5708

				_3.67321e_6 2 o. rdf 90 6.12303e_17 (COS 90°の値)

```
(Circular Functions)
   1 o.y ⇔ sin y        _1 o.y ⇔ arcsin y
   2 o.y ⇔ cos y        _2 o.y ⇔ arccos y
   3 o.y ⇔ tan y        _3 o.y ⇔ arctan y
   5 o.y ⇔ sinh y       _5 o.y ⇔ arcsinh y
   6 o.y ⇔ cosh y       _6 o.y ⇔ arccosh y
   7 o.y ⇔ tanh y       _7 o.y ⇔ arctanh y
   0 o.y ⇔ %*1-y^2
   4 o.y ⇔ %:(y^2)+1    _4 o.y ⇔ %:(y^2)-1
   8 o.y ⇔ %:-1+y^2     _4 o.y ⇔ -%:-1+y^2
   9 o.y ⇔ Real(y)      _9 o.y ⇔ y
  10 o.y ⇔ Abs(y)      _10 o.y ⇔ Conj(y)
  11 o.y ⇔ Im(y)       _11 o.y ⇔ y*0j1
  12 o.y ⇔ Arg(y)      _12 o.y ⇔ ^(y*0j1)
```

１.４　複素数の計算

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
+ Conjugate 共役	＋	共役複素数を与える + 3j4 3j_4	Plus 加算 X + Y	複素数どうしの足し算 3i4 + 3j_4 6
- Negate 逆符号	－	実部と虚部の符号を逆にする - 3j4 _3j_4	Minus 減算 X - Y	複素数どうしの引算 3.5j0.5 - 1j1 2.5j_0.5
* Signum 符号 * Y	×	大きさが１で、同じ偏角の複素数を与える * 3j4 0.6j0.8	Times 乗算 X * Y	複素数どうしの掛け算 3j4 * 3j_4 25
% Reciprocal 逆数 % Y	÷	「％ｙ」は「１％ｙ」と同じ 複素数の逆数は大きさが逆数で、偏角が反対符号になる % 3j4 0.12j_0.16 3j4 * % 3j4 1 *. 3j4 5 0.927295 *. % 3j4 0.2 _0.927295	Devide-by 除算 X % Y	複素数どうしの割算 3j4 % 1j1 3.5j0.5 1j1 * 3j4 % 1j1 3j4
Magnitude 絶対値 Y		符号をとって絶対値にする 6 _5 6 5 3j4 5 複素数に対しては大きさ	Residue 剰余 X　Y	右引数を左引数で割ったときの余り 3　i.7 0 1 2 0 1 2 0 3　-i.7 0 2 1 0 2 1 0

				_3 i.7 0 _2 _1 0 _2 _1 0 _3 -i.7 0 _1 _2 0 _1 _2 0
+. Real/ Imaginary 実部／虚部 +. Y		複素数の実部と虚部を生成 +. 1j2,2j3,3j4 0 0 1 2 2 3 3 4	GCD(Or) 最大公約数 （論理和） X +. Y	左右の引数が共に０のときに０、そうでなければ１ 24 +. 60 12 0 0 1 1 +. 0 1 0 1 0 1 1 1

*. Length /angle 長さ／偏角 *. Y	∧	*. 3 3 0 *. 3j4 5 0.927295	LCM(And) 最小公倍数 （論理積） X *. Y	左右の引数が共に１のときに１、そうでなければ０ 24 *. 60 120 0 0 1 1 *. 0 1 0 1 0 0 0 1
j. Imaginary 虚数生成 j. Y		「j. y」は「０ｊ１＊ｙ」]b=.j. a=.1j1 _1j1 (b-a),(b=.*.b),:a=.*.a 0 1.5708 1.41421 2.35619 1.41421 0.785398 rdf 90 1.5708 「j. y」は y を複素平面上で９０度回転する	Complex 複素数生成 X j. Y	x j. yは「x + j. y」 3 j. 4 3j4 3 + j. 4 3j4
r. Angle 単位複素数 r. Y		r. yは「^ 0j1 * y」 r. 0 1 1 0.540302j0.841471 ^0j1*0 1 1 0.540302j0.841471 *. r. i.3 1 0 1 1 1 2 実数は単位円周上に移される *. r. 1j1 1j0 1j_1 0.367879 1 1 1 2.71828 1 虚数部に反比例して絶対値が大きくなる。	Polar 極座標表示 X r. Y	「x r. y」は「x*^0j1*y」 r. 2 _0.416147j0.909297 2 r. 2 _0.832294j1.81859

１.５　マトリクスの計算

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
u .v Deter- minant 一般行列式		「-/ .*」は正則行列の行列式 <A=.3 3$1 6 6 4 1 0 6 6 8	Dot product 一般内積	「X+/ .* Y」は内積を表す (i.5)+/ .*i.5 30 A=.>:i.3 4

		1 6 6 4 1 0 6 6 8 -/ .* A _76 「+/ .*」はパーマネント(Permanent) +/ .* A 380		B=.>:i.4 2 A +/ .* B 54 60 114 140 178 220 u，vとしてはいろいろな動詞を適用することも可能 X='LANVANAPLCADTSS']X=.5 3$X LAN VAN APL CAD TSS X */ .= 'APL' 0 0 1 0 0
%. Matrix Inverse 一般逆行列 %. Y		（行数≧列数）の行列の一般逆行列を与える。 %.2 2$1 0 0 2 1 0 0 0.5 M=.3 2$1 0 0 1 2 1%3 %.M 1 _1 1 _1 2.5 0.5	Matrix Divide 行列の割算 X %. Y	Xがベクトルで YがXの次数に等しい正則行列ならば連立方程式の解を与える。またYの行数が列数より大きいときは、最小２乗解。 1 4 %. 2 2$1 0 0 2 1 2 1 4 2 %. M _1 10
/ Insert （副詞）	／		Table 一般外積 X u/ Y	「X */ Y」はXとYの外積 uとしては他のいろいろな動詞を適用することも可能 1 2 3 */ 4 5 6 7 4 5 6 7 8 10 12 14 12 15 18 21 1 2 3 +/ 4 5 6 7 5 6 7 8 6 7 8 9 7 8 9 10

128!: ＱＲ分解		「128!:0」がＱＲ分解 「128!:1」はYを上三角行列に変換する (128!:0)2 2$1 0 0 2 1 0 | 1 0 0 1 | 0 2 (128!:1)2 2$1 0 0 2 1 0 0 0.5	（無し）	

１.６　乱数と素数

用　法	APL	結果と例（単項）	用　法	結果と例（二項）

? Roll 乱数 ? Y		「？ｙ」はＯからｙ－１までの整数乱数を生成する。ｙは１より大きな整数でないと[domain error]になる。 　? 10 10 10 10 5 6 0 3 「９！：１」はランダムシードをセットする。	Deal 非重複乱数 X ? Y	「ｘ？ｙ」は重複を許さずにｉ．ｙからｘ個の整数乱数を生成する。 　5 ? 5 0 2 1 4 3 　3 3 $? 9 0 3 6 5 8 7 4 0 5
?. Roll 乱数 ?. Y		シードを固定して乱数発生 　? 5 5 5 5 5 4 1 1 4 3 　? 5 5 5 5 5 3 3 0 3 4 　?. 5 5 5 5 5 0 3 2 2 1 　?. 5 5 5 5 5 0 3 2 2 1	Deal 非重複乱数 X ?. Y	シード固定で非重複乱数 　6 ? 6 5 1 2 4 3 0 　6 ? 6 1 5 0 4 3 2 　6 ?. 6 5 1 2 4 3 0 　6 ?. 6 5 1 2 4 3 0
p: Primes 素数 p: Y		右引数で指定した順位の素数を与える 　p: 0 1 2 3 4 2 3 5 7 11	（無し）	
q: Prime factor 素因数分解 q: Y		右引数の整数の素因数分解 　q: 12 2 2 3 　q: 700 2 2 5 5 7	Prime Exponents 素数の位置 X q: Y	素因数分解した素数の位置と個数を与える 　9 q: 700 2 0 2 1 0 0 0 0 0 　p: i.9 2 3 5 7 11 13 17 19 21

１．７　微分、テイラー展開、多項式

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
d. Derivative 微分 (接続詞) u d.n		「ｕｄ．ｎ」は関数ｕのｎ階の数値微分を行う ランクはＯに固定されている 　*: d.1 +: 　*: d.1 x=.1 2 3 2 4 6		
D. Derivative 微分 (接続詞) u D.n		「ｕＤ．ｎ」も関数ｕのｎ階の数値微分を行う ランクはＯに固定されてない 　(cube=.^&3"0)y=.2 3 4 8 27 64 　cube D.1 y 12 27 48 　cube D.2 y 12 18 24		
D: Secant				cube=.^&3"0 　1 cube D:1 y=.2 3 4

Slope 平均変化率 (接続詞) u D:n				19 37 61 1 0.1 0.01 cube D:1/y 19 37 61 12.61 27.91 49.21 12.0601 27.0901 46.1201
T. Taylor approxi-mation (接続詞) u T.n		「u T.n」は関数ｕのｎ項までのテイラー展開で近似した値 ^T.5 i.3 1 2.70833 7 ^T.30 i.3 1 2.71828 7.38906 ^ i.3 1 2.71828 7.38906	（無し）	
t. Taylor co-efficient (副詞) u t. Y		「(ｕｔ．)ｙ」は関数ｕのテイラー展開のｙでの係数 sin=.1&o.[cos=.2&o. sin t. i.5 0 1 0 _0.166667 0 cos t. i.5 1 0 _0.5 0 0.0416667	Taylor co-efficient (副詞) X ｕｔ．Y	「ｘ(ｕｔ．)ｙ」はｘ＾ｙと(ｕｔ．)ｙとの積

t: Weighted Taylor co-efficient (副詞) u t: Y		階乗により重みづけられたテイラー展開。「u t: Y」はは「(!Y)* t. Y」と同じ sin t: i.5 0 1 0 _1 0 (!i.5)*sin t: i.5 0 1 0 _1 0 cos t: i.5 1 0 _1 0 1		
p.		（無し）	Polynomial 多項式 X ｐ．Y	左引数で与えた係数の多項式の右引数での値を与える x=.1 5 0 4 [y=._1 0 2 x p. y _8 1 43 (1+5x+4x3)のx=_1,0,2の値

１.８　階乗と置換

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
! Factorial 階乗 ! Y	!	「！ｙ」は１，２，‥‥，ｙの積(階乗)を与える。 「＊/@(＞：@ｉ.)”０」としても同じである ! 3 4 5 6 24 120 「！@＜：ｙ」はガンマ関数 !@<: 5 24 !@<: 4.5	Out-Of ２項係数 X ! Y	「ｘ！ｙ」はｙ個の中からｘ個取り出す組合せ総数 2 ! 8 28]a=.!2 2]b=.!8-2 720]c=.!8

		11.6317 整数値以外でも算出できる。		40320 c % a*b 28
A. Anagram Index A. Y		数列や文字列の置換のインデックスを与える。]y=.(i.6)A.0 1 2 0 1 2 0 2 1 1 0 2 1 2 0 2 0 1 2 1 0 A. y 0 1 2 3 4 5	Anagram 辞書式順序 X A. Y	Xで与えたインデックスに対応するYの置換を与える (i.6)A.'abcd' abcd abdc acbd acdb adbc adcb 最初から６通りの配列が得られる。

C. Cycle direct C. Y		並べ変えの指標を与える y=.1 2 0 4 3 C. y 2 0 1 4 3 /: y 2 0 1 4 3 y C. 'abcde' bcaed 'abcde'/:(/:y) bcaed	Permute X C. Y	Xで与えた指標に従って、Yを並べ変える。 y=.1 2 0 4 3 y C. 10 11 12 13 14 11 12 10 14 13 10 11 12 13 14 /: /:y 11 12 10 14 13

１.９　接続詞の「..」と「.:」

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
..			Even （接続詞） u .. v	２つの動詞を接続して１つの動詞になる。 「-:@(u+u&v)」と同じ (*: ; *:&+:)4 16 64 -:@(*:+*:&+:)4 40 *: ..+: 4 40 1 2 3 4 5&p. 2 _2 129 64 1 2 3 4 5&p. ..- 2 93 1 0 3 0 5&p. 2 93
.:			Odd （接続詞） u .: v	２つの動詞を接続して１つの動詞になる。 「-:@(u-u&v)」と同じ -:@(*:-*:&+:)4 _24

				*: .:+: 4 _24

第２章　論理演算とブール代数

２.１　論理演算

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
-: (Halve) (1.1)			Match 一致	左引数と右引数とがマッチしていれば１、そうでなければ０ y -: y=.i.3 3 1
～.(副詞) Nub 重複排除		Ｙの重複要素を排除 ～. 1 1 2 2 3 3 1 2 3	(無し)	
～: Nubsieve 重複指示 ～: Y	≠	～:1 1 2 2 3 3 1 0 1 0 1 0	Not equal 不等 X ～: Y	ＸとＹが等しければ０、そうでなければ１ 20 30 40 ～:40 30 20 1 0 1
= Self Classify 自己分類 = Y	＝	重複を分類して判別し、１で重複、０で非重複を表示する a=.3 3$'abcdefghi' abc def ghi = a 1 0 0 0 1 0 0 0 1 = i.3 1 0 0 0 1 0 0 0 1 （３×３の単位行列を生成）	Equal 等しい X = Y	ＸとＹが等しければ１、そうでなければ０ 20 30 40 = 40 30 20 0 1 0
e. Raze(in) 部分所属 e. Y	ε	Ｙの各要素を照合してブール数で表示する。 e. i.5 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1	Member(in) 要素の所属 X e. Y	左引数の要素が右引数の中に含まれていれば１、そうでなければ０を返す 8 7 6 7 9 e. 7 8 9 7 1 1 0 1 1

E.		（無し）	Memder of Interval パターンの所属 X E. Y	右引数の中に左引数のパターンが含まれていれば、始まる位置に１を返す。そうでなければ０を返す。 'co' E. 'cocoa' 1 0 1 0 0

i. (Integers) (3.1)	⍳		Index of First インデクス X i. Y	左引数の要素の右引数における最初の位置を返す。 7 8 9 7 i. 8 7 6 7 9 1 0 4 0 2
i:		（無し）	Index of Last インデクス X i: Y	左引数の要素の右引数における最後の位置を返す。 7 8 9 7 i: 8 7 6 7 9 1 3 4 3 2 y i. y='sundrug' 0 1 2 3 4 1 6 y i: y 0 5 2 3 4 5 6
< (Box) (3.7) < Y	＜	< i.5 0 1 2 3 4	Less than 小さい X < Y	左引数が右引数より小さければ１、そうでなければ０ 20 30 40 < 40 30 20 1 0 0
> (Open) (3.7) > Y	＞	> < i.5 0 1 2 3 4	Larger than 大きい Y > Y	左引数が右引数より大きければ１、そうでなければ０ 20 30 40 > 40 30 20 0 0 1
<: Decreament (1.1)	≦		Lesser or equal 小さいか 等しい X <: Y	左引数が右引数より小さいか等しければ１、そうでなければ０ 20 30 40 <: 40 30 20 1 1 0
>: Increament (1.1)	≧		Larger or equal 大きいか 等しい X >: Y	左引数が右引数より大きいか等しければ１、そうでなければ０ 20 30 40 >: 40 30 20 0 1 1

２.２　ブール代数

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
*. Length /angle (1.4)	∧		LCM(And) 最小公倍数 （論理積） X *. Y	ＸとＹの最小公倍数 24 *. 60 120 左右の引数が共に１のときに１、そうでなければ０ 0 0 1 1 *. 0 1 0 1 0 0 0 1
+. Real /Imaginal 実部／虚部 (1.4)	∨		GCD(Or) 最大公約数 （論理和） X +. Y	ＸとＹの最大公約数 24 +. 60 12 左右の引数が共に０のときに０、そうでなければ１ 0 0 1 1 +. 0 1 0 1

				0 1 1 1
*: (Square) 平方 (1.1)	～∧		Not-And 否定論理積 X *: Y	左右の引数が共に１のときに０、そうでなければ１ 0 0 1 1 *: 0 1 0 1 1 1 1 0
+: (Double) ２倍 (1.1)	～∨		Not-Or 否定論理和 X +: Y	左右の引数が共に０のときに１、そうでなければ０ 0 0 1 1 +: 0 1 0 1 1 0 0 0
-. (Not) 否定・補数 -. Y	～	右引数が０なら１、１なら０ -. 1 0 0 1 -. 0.6 0.4	Less	左の引数から右引数の要素を除いたものを出力する (i.5) -. 1 3 0 2 4 (i.5)(-.@e.#[)1 3 0 2 4
b.(副詞) Basic Characteristic (5.6) u b.n		「u b.0」はランクを表示する * b.0 0 0 0 「u b._1」は逆関数を表示する *: b._1 %: 「u b.1」は “identity function” ^ b.1 $&1@(}.@$) (^ b.1) i.4 5 10 10	Boolean	数値をブール数に変換する (+./～,.+:/～) 1 0 1 1 0 0 1 0 0 1 (*./～,.*:/～) 1 0 1 0 0 1 0 0 1 1 <"2|:7 8 1 14 b./～0 1 ┌─┬─┬─┬─┐ │1 1│0 0│1 0│0 1│ │1 0│0 1│0 0│1 1│ └─┴─┴─┴─┘

２.３　ｎ進化、１０進化

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
#. Base-2 ２進数 #. Y		２進数で与えた右引数を１０進数に変換する #. 1 1 1 1 15 2 #. 1 1 1 1 15	Base １０進化 X #. Y	左引数を底の加重合計値 10 #. 1 2 3 123 60 #. 1 30 20 5420 １時間３０分２０秒の秒数 2 #. 4 5 6 32 (4*2^2)+(5*2)+6 32 (4x2+5x+6)の「x=2」の値 0 #.＼ 5 6 0 0 0 1 #.＼ 5 6 5 6 1(#."0)5 6 5 6 2 #.＼ 5 6 16 2(#."1)5 6

				16 3 #.＼ 5 6 (null)
#: Antibase 2 １０進数の ２進化 #: Y		１０進数で与えた右引数を ２進数に変換する #: i.8 0 0 0 ····· 0 0 0 1 ····· 1 0 1 0 ····· 2 0 1 1 ····· 3 1 0 0 ····· 4 1 0 1 ····· 5 1 1 0 ····· 6 1 1 1 ····· 7	Antibase ｎ進化 X #: Y	１０進数で与えた右引数の 左引数の底による進数表示 24 60 60 #: 5420 1 30 20 (秒の時間・分・秒へ変換) 4 4 #: i.6 0 0 0 1 0 2 0 3 1 0 1 1 (４進数表示)

第３章　アレイの各種の変形

３.１　アレイの形と変形

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
i. (Integers) 整数生成 i. Y	⍳	０からＹ－１までの整数列を生成する。 　　i.4 0 1 2 3 　　i. ¯4 3 2 1 0 負の整数なら降順の整数列 　　i.2 3 0 1 2 3 4 5 テーブルの形でも生成できる	Index of インデクス X i. Y	左引数で指定した左引数の中で最初に表われるインデクスを返す。 　　7 8 9 7 i. 8 7 9 7 9 1 0 2 0 2 　　7 8 9 7 i. 8 7 9 6 9 1 0 2 4 2 左引数にない場合には＃ｘを返す。
$ Shape of 形 $ Y	⍴	Ｙの各ランクの要素数を返す 　　]S=. i.3 4 0 1 2 3 4 5 6 7 8 9 10 11 　　$ S 3 4	shape 変形 X $ Y	Ｘの形にＹを変形する。 　　S=. i.3 　　2 2 4 $ S 0 1 2 3 4 5 6 7 8 9 10 11 0 1 2 3
# Tally アイテム数 # Y		アイテム（引き数より１つランクの低いセル）の数 　　# i.3 4 3 　　# 4 1 アトムのアイテムはそれ自身	Copy 複写 (3.3)	
, Ravel リスト化 , Y	,	右引数をリストに変形する。 　　]S=.i.2 3 0 1 2 3 4 5 　　, S 0 1 2 3 4 5 　　$, 3 1 アトムもリストに変形される	Append 連結 X , Y	右引数を左引数の最大ランクの方向に連結する。 左引数がリストなら横に、テーブルなら下に連結する 　　](S=.i.2 3),6 7 8 0 1 2 3 4 5 6 7 8 　　S,6 7 0 1 2 3 4 5 6 7 0 要素数が不足なら０を付加して成形される。
,. Ravel Items テーブル化		右引数をテーブル化する。 　　]a=. ,.3 3 見かけはアトムのようだが 　　$ a	Stitch 縦連結 X ,. Y	「ｘ,.ｙ」は「,.ｙ」とｘを「,」で連結したのと同じ 　　1 3 5 ,. i.3 1 0 3 1

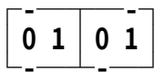
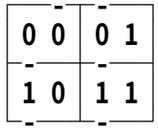
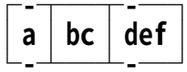
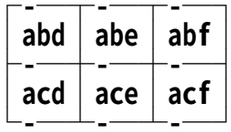
,. Y		1 1 ,. i.3 0 1 2 $,. i.2 3 4 2 12		5 2 5 ,. i.3 2 5 0 1 5 2 3 5 4 5
,: itemize アイテム化 ,: Y		ランクを１つ増やした高次のアレイを作る。]S=.,: i.2 3 0 1 2 3 4 5 見かけはテーブルのようだが $ S 1 2 3 $,: 3 1	Laminate 層連結 X ,: Y	セルを重ねて、１つ高いランクになるように連結する]S=.(i.3),:i.2 3 0 1 2 0 0 0 0 1 2 3 4 5 $ S 2 2 3

３.２　ランク

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
" Rank ランク u"n Y		動詞(ｕ)の作用するランクセルのナンバーを指定する。 y=.i.2 3 4 +/"1 y 6 22 38 54 70 86　(行方向の合計) （ランク１のアイテムの和） +/"2 y 12 15 18 21 48 51 54 57 (列方向の合計) （ランク１のアイテムの和） +/"3 y 12 14 16 18 20 22 24 26 28 30 32 34 省略した場合は最大ランク ＡＰＬでは軸を指定するが、Ｊではランクという概念が採用され、微妙に差異がある。	Rank ランク X u"n Y	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 (Ｘ，Ｙと同じである) X ,"0 X 7 7 8 8 9 9

３.３　取りと落し

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
# Tally アイテム数 (3.1)			Copy 複写 X # Y	右引数を左引数で指定した数だけコピーする。 5 # 1 1 1 1 1 1 1 2 3 # 4 5 6 4 5 5 6 6 6 0 0 1 0 1 # 1 2 3 4 5 3 5

				１の所だけに圧縮する E=.=/～i.3(単位行列) ,E #"1 A=.i.3 3 0 4 8 行列の圧縮(Ａの対角要素) ＡＰＬの圧縮(／)に対応。
{ Catalogue		ボックスで与えられた要素の全ての組合せを与える。 アトムにはボックスと同じ。 0 1 ; 0 1  { 0 1 ; 0 1 ]C='a';'bc';'def'  { C 	{ From 選択 X {Y	左引数で指定したアイテムを取り出す。]Y=.i.4 5 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 0 { Y 0 1 2 3 4 ０行の取り出し 0 1 { Y 0 1 2 3 4 5 6 7 8 9 ０，１行の取り出し 2 3 {"1 Y 2 3 7 8 12 13 17 18 ２、３列の取り出し (<2 3;3 4){ Y 13 14 18 19 ブロックで取り出すときは２重ボックスで指定する。

{. Head 先頭取り {. Y	↑	{. A=.1 2 3 4 1 右引数の先頭アイテム(要素)を取り出す。 {. B=.i.3 4 0 1 2 3 先頭アイテム(行)の取り出し {. C=.i.2 3 4 0 1 2 3 4 5 6 7 8 9 10 11 先頭のテーブルの取り出し。	Take 取り X {. Y	Ｘが正整数のときは、Ｙの先頭からＸ個のアイテムをまた負なら末尾から|Ｘ|個のアイテムを取り出す。 2 {. A=.1 2 3 4 1 2 _3 {. A 2 3 4 2 3 {. B=.i.3 4 0 1 2 4 5 6 ２行３列のアイテム 2 2 4{. C(_ 2 _{. C) 0 1 2 3 4 5 6 7 12 13 14 15 16 17 18 19 ２枚の２行４列のアイテム
}. Behead	↓	}. A=.1 2 3 4 2 3 4	Drop 落し	Ｘが正整数のときは、Ｙの先頭からＸ個のアイテムを

先頭落し }. Y		先頭アイテム(要素)を落す。 }. B=.i.3 4 4 5 6 7 8 9 10 11 先頭のアイテム(行)を落す。 }. C=.i.2 3 4 12 13 14 15 16 17 18 19 20 21 22 23 先頭のテーブルを落す。	X }. Y	また負なら末尾から|X|個のアイテムを落す。 2 }. 1 2 3 4 8 9 10 11 _2 }. B 0 1 2 3 1 2 }. i.3 4 6 7 10 11 先頭から１行２列を落す。
{: Tail 末尾取り {: Y		右引数の最終アイテムの取リ {: 1 2 3 4 4 {: i.3 4 8 9 10 11 最終アイテム(行)の取り	（無し）	
}: Curtail 末尾落し }: Y		右引数の最終アイテムの落し }: 1 2 3 4 1 2 3 }: i.3 4 0 1 2 3 4 5 6 7 最終アイテム(行)の落し	（無し）	

３．４　修正

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
} Item Amend ｱｲﾃﾑ修正 m } Y		mで指定したＹの各アイテムに変更したものを表示する。]Y=.2 5$'abcdefghij' abcde fghij 1 0 1 0 1 } Y fbhdj 1 0 1 0 2 } Y index error Ｊでは、Ｙ自体は変更されないので Y=.1 0 1 0 1 } Y のように挿入する必要があるＡＰＬの場合には、「←」で内容が直ちに修正される。	Amend 修正 X m } Y	インデクスmで指定したＹの要素をＸに変更して表示 Y=.'abcdefghij' '*' 1 3 5 7 } Y a*c*e*g*ij 'BD' 1 3 } 'abcd' aBcD m=.0 1;1 2;2 3 12 13 14 m }i.3 4 0 12 2 3 4 5 13 7 8 9 10 14 （要素の変更はボックス）]X=.12+i.2 4 12 13 14 15 16 17 18 19 X 1 2 } i.3 4 0 1 2 3 12 13 14 15 16 17 18 19 X(<<1 2)} i.3 4 0 1 2 3 12 13 14 15 16 17 18 19 １，２行(アイテム)の変更]X=.12+i.3 2 12 13

				14 15 16 17 X 1 2 }"1 i.3 4 0 12 13 3 4 14 15 7 8 16 17 11 X(<<1 2)}"1 i.3 4 0 12 13 3 4 14 15 7 8 16 17 11 行内(１，２)の列の修正 12 13(<0 1;1 2)}i.3 4 0 12 13 3 4 12 13 7 8 9 10 11 0,1行と1,2列の個所を修正

３．５　逆順、回転とソート

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
. Reverse 逆順 . Y		アイテム（引き数より１つランクの低いセル）を逆順に . 'abcdefg' gfedcba . 3 3$'abcdefghi' ghi def abc . i.2 2 2 4 5 6 7 0 1 2 3 アイテムの順序を反対にする	Rotate 回転 X . Y	右引数のアイテムをＸ個だけ回転する。 1 . i.5 1 2 3 4 0 _1 . i.5 4 0 1 2 3 (Ｘが負なら後ろから回転) 2 . i.5 2 3 4 0 1 「1 .」を２度適用と同じ 1 2 . i.3 4 6 7 4 5 10 11 8 9 2 3 0 1 「[:1& .2& ."1」と同じ
: Transpose 転置 : Y	＼	右引数のセルを転置させる。 : i.3 4 0 4 8 1 5 9 2 6 10 3 7 11 (]; :)i.2 2 2 ┌─┬─┐ │0 1│0 4│ │2 3│2 6│ ├─┼─┤ │4 5│1 5│ │6 7│3 7│ └─┴─┘ (<1 1 0){i.2 2 2 6 (<0 1 1){ :i.2 2 2 6 Ｙの２枚目の２行１列の要素は、(：Ｙ)では１枚目の２	Transpose ２項転置 X : Y	Ｘで指定した軸が最も若い軸(０軸)となるよう転置。 1 : i.2 3 3 4 5 0 1 2 Ｘで指定した軸が０軸になるように変形されていて、(０ １）⇒（１ ０)のように軸が変換される。 $ 0 : i.2 3 4 3 4 2 (０ １ ２）⇒（１ ２ ０）のように軸が変換される。 $ 0 1 : i.2 3 4 4 2 3 (０ １ ２）⇒（２ ０ １） (<0 1) :i.4 4 0 5 10 15 対角要素が取り出される。 (<0 1 2) : i.2 3 4

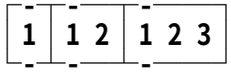
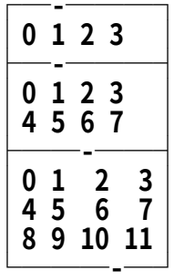
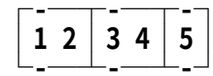
		行２列の要素になる. 　　$　：i.3 4 5 5 4 3 　：(３面４行５列のアレイ)は５面４行３列のアレイになる。		0 17 (０　０　０)と(１　１　１)の要素が取り出される。

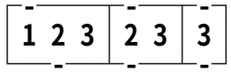
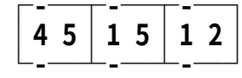
/: Grade-up 昇順 /: Y		Ｙを昇順に並べるインデクスを出力する。 　/: 23 11 13 31 12 1 4 2 0 3 最小値はインデクス１に、最大値はインデクス３にある。 　Y 74 1 26 23 33 87 34 10 66 50 20 45 　(/:Y),:/:74 33 66 1 2 0 1 2 0 テーブルのときは行単位に先頭から順番に比較したときの数値に対する指標 　/:'shimura' 6 1 2 3 5 0 4 文字の場合はアルファベット	Sort 昇順ソート X /: Y	ＸをＹで与えた指標に従って並べ替える。 　74 33 66 /: 2 1 0 66 33 74 　2 1 0 { 74 33 66 66 33 74 　Y=.23 11 13 31 12 　Y /: Y 11 12 13 23 31 　/:～ Y 11 12 13 23 31 　(/:Y){ Y 11 12 13 23 31 いずれも昇順に並べ替える 　a='shimura' 　b='APLJ' 　/:～ a ahimrsu 　a /: b smih 　b /: a index error
＼: Grade-down 降順 ＼: Y		Ｙを降順に並べるインデクスを出力する。 　＼: 23 11 13 31 12 3 0 2 4 1 最大値はインデクス３に、最小値はインデクス１にある。	Sort 降順ソート X ＼: Y	ＸをＹで与えた指標に従って並べ替える。 　74 33 66 ＼: 2 1 0 74 33 66 　＼:～ Y=.23 11 13 31 12 31 23 13 12 11 　＼:～a='shimura' usrmiha 　a ＼: b hims 　Y 74 1 26 23 33 87 34 10 66 50 20 45 　＼: Y 0 2 1 　＼:～ Y 74 1 26 23 66 50 20 45 33 87 34 10

３．６　データの融合

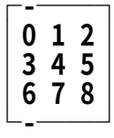
用　法	APL	結果と例（単項）	用　法	結果と例（二項）
/ Insert 挿入 u / Y m / Y （副詞）	／	動詞(ｕ)の場合は、アイテム間にｕを挿入した演算と同等 ジェランド(m)の場合は、ｍの各要素を右引数の各アイテムに指定順に挿入する。 　+/ i.5 10 　0+1+2+3+4	Table 一般外積 (1.5)	

		10
		   +/ i.2 5
		5 7 9 11 13
		   +/"1 i.2 5
		10 35
		テーブル等の一般アレイにはランクで作用方向を指定する
		   +`*/ i.5
		14
		   0+1*2+3*4
		14		

＼(\)
Prefix
前から
u＼ Y
（副詞）
		動詞ｕを１から＃Ｙまで逐次累加した組合せに作用させる
		   <＼ 1 2 3
		 
		   +/＼ 1 2 3
		1 3 6
		   ,.<＼ i.3 4
		 
			Infix
			中から
			X u＼ Y
				左引数を区切りに前から逐次取り出した組合せに作用
				Ｘが正ならオーバーラップさせ、負のときは重複なし
				   2 <＼ 1 2 3 4 5
				
				  _2 <＼ 1 2 3 4 5
				 
				   2 +/＼ 1 2 3 4 5
				3 5 7 9
				  _2 +/＼ 1 2 3 4 5
				3 7 5
				（応用：移動平均）
				   mave=.+/＼%[
				   3 mave 1 2 3 4 5
				2 3 4

＼.
Surfix
後から
u＼. Y
（副詞）
		動詞ｕを１から＃Ｙまで前から減少させた組合せに作用
		   <＼. 1 2 3
		 
		   +/＼. 1 2 3
		6 5 3
			Outfix
			外から
			X u＼. Y
				3 <＼. 1 2 3 4 5
				 
				  3 +/＼. 1 2 3 4 5
				9 6 3

３.７　ボックスによる囲みとほぐし

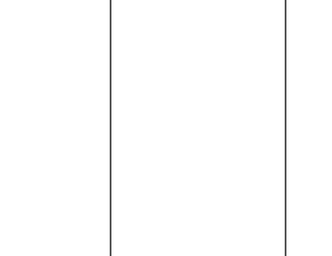
用　法	APL	結果と例（単項）	用　法	結果と例（二項）
< Box < Y		右引数をボックスで囲む。]B=.< i.3 3 	Less than 小さい X < Y （2.1）	
> Open		ボックスを開く。 > B	Larger than	

> Y		0 1 2 3 4 5 6 7 8 （Ｊでは、ボックスの状態で演算しないので、ボックスを開いて計算する必要がある。ボックスの中で演算させるにはレベル(Ｌ：)を使用する）	大きい X > Y (2.1)	
; Raze ほぐし ; Y		主軸に沿って右引数をほぐす ; i.2 3 0 1 2 3 4 5 ;/ i.2 3 ┌─┬─┐ │0 1 2│3 4 5│ └─┴─┘]Y=.;:'I like J' ┌─┬────┬─┐ │I│like│J│ └─┴────┴─┘ ; Y IlikeJ ; <i.2 3 0 1 2 3 4 5	Link 結合 X ; Y	左右の引数をボックスの形で結合する。 0 2 4 6 ; 1 3 5 7 ┌─┬─┐ │0 2 4 6│1 3 5 7│ └─┴─┘ 「(＜X)，(＜Y)」と同じ (0 2 4 6); 'abcdef' ┌─┬──┐ │0 2 4 6│abcdef│ └─┴──┘ 数値と文字列の結合も可能 ; (0 2 4 6); 'abcdef' domain error 数と文字の混合はほぐせず ┌─┬─┐ │┌─┐│1 3 5 7│ │└─┘│ │ └─┴─┘ 0 2 4 6 ;<1 3 5 7 ┌─┬─┐ │0 2 4 6│1 3 5 7│ └─┴─┘ Yがボックスのときは 「(＜X)，Y」と同じ！
L. Level L. Y		右引数のボックスのレベルの深さを与える。Yが空又はオープン形のときは0である L. i.5 0 L. 1 2;3 4 1]Y=.(<0 2 4 6); 1 3 5 7 ┌─┬─┐ │┌─┐│1 3 5 7│ │└─┘│ │ └─┴─┘ L. Y 2	（無し）	
L: Level		（無し）	Level 副詞 u L:n Y	動詞ｕをＬ：ｎというレベルのボックス内で演算する +: L:0(123;456) ┌─┬─┐

				┌─┬─┐ 246 912 └─┴─┘ $ L:0(123;456) ┌─┬─┐ └─┴─┘ $ L:1(123;456) 2
{:: Map {:: Y		ボックスへのパスを表示 {:: Y=.2 3;(2;3);i.2 3 ┌────┬───────┬───────┬───┐ │┌─┐ │┌─────┐ │┌─────┐ │┌─┐ │ │└0┘ ││┌─┬─┐ ││┌─┬─┐ │└2┘ │ │┌─┐ ││└0┴1┘ ││└1┴1┘ │┌─┐ │ │└─┘ ││┌─┬─┐ ││┌─┬─┐ │└─┘ │ │ ││└─┴─┘ ││└─┴─┘ │ │ └────┴───────┴───────┴───┘	Fetch X {:: Y	左引数で与えた指標に従い右引数のサブアレイの要素を取り出す。 (2;1 2){:: Y 5 指標２のボックス(i.2 3)の(1 2) の要素を取り出す (1;1 2){:: 5;i.2 3 4 20 21 22 23

３．８　区切り(Cut)と斜め(Oblique)

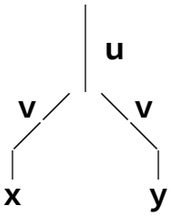
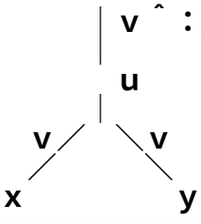
用　法	APL	結果と例（単項）	用　法	結果と例（二項）
;. Cut 区切り (接続詞) u ;.n Y (nはカットの型)		右引数のフレット（先頭又は末尾のアイテム）で区切る。];.0 L='hand in hand' dnah ni dnah];.0 D=.i.2 2 3 2 1 0　　(逆順にする)];1. L hand in hand <;1.(3 2$i.4) ┌─┬─┐ │0 1│0 1│ │2 3│　│ └─┴─┘ 先頭のフレット(0 1や'h')が現れた個所で区切る。 <;._1(3 2$i.4) ┌─┬─┐ │2 3│　│ └─┴─┘ 先頭のフレットを除き区切る <;.2(3 2$i.4) ┌─┬─┐ │0 1│2 3│ │　│0 1│ └─┴─┘];.2 L hand in hand 末尾のフレット(0 1や'd')が現れた個所で区切る。	Cut 区切り (接続詞) X u;.n Y	ｎ＝０のとき左引数で指定した個数の要素を取り出すｎ＝１，２のときはブール数の１の個所でカットする 2 2];.0 i.3 2 0 1 2 3 2 1];.0 i.3 2 0 2 1 0 1 <;.1 i.3 2 ┌─┬─┐ │0 1│4 5│ │2 3│　│ └─┴─┘ 「<;.1」は先頭から１が現れるごとに区切って出力。 1 0 1 <;._1 i.3 2 ┌─┬─┐ │2 3│　│ └─┴─┘ 「<;.1」の先頭を削除する 0 1 0<;.2 i.3 2 ┌─┐ │0 1│ │2 3│ └─┘ 「<;.2」は末尾から１が現れるごとに区切って出力。 0 1 0];._2 i.3 2 0 1 「<;.2」の末尾を削除する

		];_2. L han in han 末尾のフレットを除き区切る];.3 i.3 0 1 2 1 2 0 2 0 0　　（モザイク模様） 一般に「<;.3 y(リスト)」は「(#y)<;.3 y」と同じ演算。 「<;.3 D(3 2 のテーブル)」は「2 2 <;.3 D」と同じ演算 「<;._3 D」についても同様		n=3 は左引数の形でカット 2 2 <;.3 i.3 2  「<;._3」の結果は上記の２行１列の部分になる.
/. Oblique 斜め u /. Y m /. Y (接続詞)		動詞(ｕ)の場合は、テーブルの形の右引数に対して右上から左下に斜めに作用させる。 ジェランド(m)の場合は、mの各要素を右引数の各アイテムに逐次作用させる。]Y=.i.3 4 0 1 2 3 4 5 6 7 8 9 10 11 +//. Y 0 5 15 18 17 11 *//. Y 0 4 80 162 70 11 （応用例） +//. 1 2 1*/1 3 3 1 1 5 10 10 5 1 ２，３の２項係数から５の場合の２項係数が得られる （ジェランドの場合） !`*: /. 3 4 5 6 16 120 順に(！３),(＊：４),(！５)を求めている	Key キー X u/. Y X m/. Y	動詞(ｕ)の場合は、Ｘをキーとして右引数の要素をグループ別に一縛りにする。 ジェランド(m)の場合は、mの各要素を逐次適用する x=.1 2 3 1 3 2 1 x </. i.7  x +//. i.7 9 6 6 x +:/. i.7 0 6 12 2 10 0 4 8 0 cly=.[:<"_1[:>=@[;"1] box=.[:({.#{:)&.>cly sum=.[:+/&>box double=.[:+:&>box x box i.7  x sum i.7 9 6 6 x double i.7 0 6 12 2 10 0 4 8 0 （ジェランドの場合） index=.[:+/=*[:i.#@〜. index 3 2 3 0 1 0 3 2 3 +:`-:/. 2 3 4 4 8 1.5 0 0 1 0 +:`-:/. 2 3 4 4 8 1.5 0

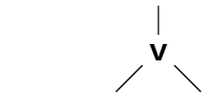
				index 3 2 8 0 1 2 3 2 8 +:`-:/. 2 3 4 4 1.5 8 インデクス順に動詞が作用

第４章　関数の合成

４.１　接続詞の「&」と「@」

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
& Bond ボンド （接続詞） u&n Y n&v Y Compose u&v Y		動詞と数(名詞)、又は　数と動詞を接続して新たな動詞を作る。さらに　２つの動詞を連結し動詞を作る。２つの動詞が単項ときは「u@v」と同じ働きをする。 %&2 y=.1 2 3 4 0.5 1 1.5 2（yを２で割る） 2&% y 2 1 0.6666667 0.5 （２をｙで割っている） +:&>: 5 12 +:@>: 5 12 2*1+5 12	Compose コンポーズ X u&v Y	二項の動詞ｕと単項の動詞ｖを＆で連結した場合は、その働きに注意が必要。 つまり「x(u＆v)y」は「(v x)u(v y)」のように作用する。 100 -&+: 40 120 (+:100)-(+:40) 120 
&. Under アンダー （接続詞） u&.v Y		「ｕ＆．ｖ」は、「ｕ＆ｖ」の後にｖの逆演算を行なった結果を返す。 +:&.>: 5 11 <:+:&>: 5 11 (>:^:_1)&+:&>: 5 11 >:^:_1 <:	Under アンダー X u&.v Y	「&」で結合させた演算の後で右動詞ｖの逆演算を行なった結果を返す。 100 -&.+: 40 60 -:(+:100)-(+:40) 60 +:^:_1(+:100)-(+:40) 60 
&: Appose アポーズ u&:v Y		ランクが無限であることを除き「&」と同じ働きをする。 +:&:>: 5 12	Appose アポーズ X u&:v Y	「&」とほぼ同じ働き 100 -&:+: 40 120 (+:100)-(+:40) 120

@ Atop アトップ （接続詞） u@v Y		２つの動詞を関数合成により連結して動詞を作る。２つの動詞が単項ならば「ｕ＆ｖ」と同じ働きをする。 +:&>: 5 12 +:@>: 5	Atop アトップ （接続詞） X u@v Y	単項の動詞ｕと二項の動詞ｖを@で連結した場合は、その働きに注意が必要。 つまり「x(u@v)y」は「u(x v y)」のように作用する。 u

		12
		  %&2 i.5
		0 0.5 1 1.5 2
		  %@2 i.5
		domain error
		名詞との連結の不可能な点が＆と異なる。
				
				
				  3 |@- 7
				4
				  |(3-7)
				4

@:
At
アット
（接続詞）
u@:ｖ Y
		ランクが無限であることを除き「@」と同じ働きをする。
		  +:@:>: 5
		12
		「@」は接続される直前の右のランクを継承するが@：は継承しない。@でうまく作動しない場合は@：に変えてみるとよい場合がある。
			At
			アット
			（接続詞）
			X u@:ｖ Y
				「@」とほぼ同じ働き
				  100 +:@:- 40
				120
				  3 |@:- 7
				4
				  3 |&:- 7
				_1
				  (-3)|(-7)
				_1

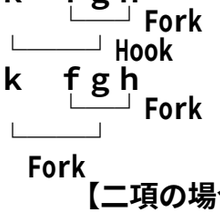
４.２　副詞のチルド「～」

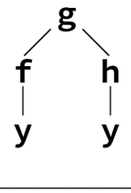
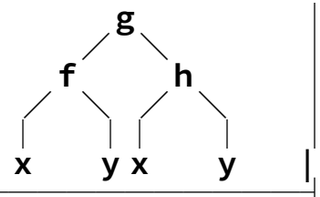
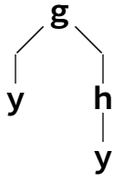
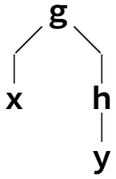
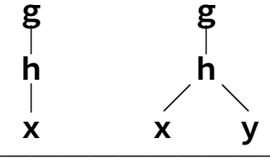
用　法	APL	結果と例（単項）	用　法	結果と例（二項）

～
Reflexive
両側化
（接続詞）
u～ Y
		二項動詞に「～」を付加すると、Ｙを左右の引数として両側演算を行なう。
		「u～ y」は「y ｕ y」と同じ
			Passive
			交換
			（接続詞）
			X u～ Y
				「ｕ～」は左右の引数を反対にして演算を行なう。
				  1 2 3 -～ 4 5 6
				3 3 3
				  (4 5 6)-(1 2 3)
				3 3 3

～
Evoke
呼出し
（接続詞）
'm'～ Y
		ユーザー定義の動詞(代動詞)を名前で呼び出してその動詞を実行させる。
		「'm'～ y」は「m y」と同じ結果を与える。
		  sum=.+/
		  'sum'～ i.5
		10
			（無し）	

４.３　フォークとフック

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
Fork フォーク (３連動詞) (ｆｇｈ)Y		Ｊの動詞(演算子)の連結では３つの動詞の結合(フォーク)が最優先される。ただ２つだけの場合はフックになる。 (+/%#)a=.1 2 3 4 5 3 【単項の場合】	Fork フォーク X(ｆｇｈ)Y	5 (+*-) 3 16 (5+3)*(5-3) 16 ４つ以上の連結の場合には  【二項の場合】

				
Hook フック (２連動詞) (ｇｈ)Y		二項動詞ｇと単項動詞ｈでは「(ｇｈ)y」は「ｙｇｈ(y)」のように演算する。 (*-)3 _9 3 *(-3) _9 	Hook フック X(ｇｈ)Y	単項の場合と同様で「x(ｇｈ)y」は「xｇｈ(y)」と演算する。 5(*-)3 _15 5 *(-3) _15 
[: Cap キャップ （動詞） ([:ｇｈ)Y		２連動詞の左端の単項動詞をフックではなく働かせたいときさらにその左端に付けて、「([:gh)y」又は「x([:gh)y」う形で演算させ「g{h(y)}」,「g(xhy)」と同じ結果を与える。 ([:*-)3 _1 *(-3) (片側の場合) _1 5([:*-)3 1 (両側の場合) *(5-3) 1 【片側】 【両側】 	Cap キャップ X [: Y	単項又は二項関数の意図しない用法を排除する。 abs=.| : [: abs _4 0 5 4 0 5 3 | _4 0 5 4 0 5 3 abs _4 0 5 valence error |@] _4 0 5 4 0 5 これで単項に限定できたか思えるが、 3 |@] _4 0 5 4 0 5 のように左右に引数を入れてもエラーにならない。

４.４　ジェランドとコントロール機能

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
` Tie (Gerund) （接続詞）  u`v/ u`v/. u`m@. n`v@. u`v@.		複数個の動詞を連結して動名詞(ジェランド)を作る。スラッシュ(/)やオブリーク(/.)アジェンダ(@.)と共に用いる +`*/ i.6 29 0+1*2+3*4+5 29 +`%/ 3 1 4 3.25	（無し）	

		3+1%4 3.25 （演算子を交互に挿入する） +:`*/. _2 3 _4 1 （演算子を交互に作用する） m=.+:`-:`*:`%: v=.4&|@<. v i.5 0 1 2 3 0 m@.v i.5 0 0.5 4 1.73205 8		
`: Evoke Gerund  m`:0 m`:3 m`:6		+:`-:`*:`:0(1 2 3) 2 4 6 0.5 1 1.5 1 4 9 （演算子を逐一作用させる） +`*`:3 i.5 14 交互に演算する「+`*/ i.5」と同じ(挿入) (+`*`:6)5 6 (+*)5 6 (Hook) (+`*`-`:6)5 _25 (+*-)5 _25 (Fork) つまり「m1`……mk`:6)」はトレインを作る。	（無し）	
$:		（無し）	Self Reference 自己参照	フレーズの生じた結果を代理して受け、左の連結詞や文の実行が完了したときに停止する(再帰呼出し) 1:`(]*$:@<:)@.* 5 120 「5*4*3*2*1」を計算する
@. Agenda アジェンダ m@.v Y		ジェランドmを実行させる条件文を記述するときに用いる m=.+:`-:`*:`%: v=.4&|@<. v i.5 0 1 2 3 0 m@.v i.5 0 0.5 4 1.73205 8	Agenda アジェンダ X m@.v Y	片側形の場合と同様 2 +`^@.< 3 8 ２＜３は真(１)なので、^の方「２ ^ ３」を実行する 3 +`^@.< 2 5 「３＋２」の方を実行する
^: Power 反復 （接続詞） u^:n Y		動詞ｕをｎ回実行する。 ｎは一般アレイでも構わない >:^:3 i.6 3 4 5 6 7 8 >:^:(i.3) i.6 0 1 2 3 4 5	Power 反復 X u^:n Y	片側形と同じく反復演算 3 *^:3 i.5 0 3 27 54 81 108 3 *^:(i.5) 3 3 9 27 81 243

		1 2 3 4 5 6 2 3 4 5 6 7 %:^:_1 i.5 0 1 4 9 16 「u ^:_1」はuの逆演算で 「*: i.5」と同じである。		
:: Adverse （接続詞） u : : v		「u : : v」の結果はエラーが無ければuを、エラーのときは停止せずにvを実行する。	（無し）	

第５章　制御構文・システム関数・ユーティリティ

５.１　明示的定義

```
  ｖｄ＝. ４  ：０    二項動詞の定義(「４」と「コロン(：)」の間にはスペースが必要)
  ｖｍ＝. ３  ：０    単項動詞の定義
  ｃｊ＝. ２  ：０    接続詞の定義
  ａｄ＝. １  ：０    副詞の定義
  ｎｏ＝. ０  ：０    名詞の定義
```

文字列として記述して置いてから、さらに品詞を定義することもできる。

```
  ｖｄ＝. ４  ：ｓ    二項動詞
  ｖｍ＝. ３  ：ｓ    単項動詞
  ｃｊ＝. ２  ：ｓ    接続詞
  ａｄ＝. １  ：ｓ    副詞
  ｎｏ＝. ０  ：ｓ    名詞
```

【明示的定義の例】

```
   mean=.3 :0
(+/y.)%#y.
)
   mean 1+i.5
3
```

「＝.」は局所定義で、大局定義は「＝:」
「ｙ.」は右引数用で左引数には「ｘ.」を使う。
「) 」は定義の終了を示す。
「i.5」は「0 1 2 3 4」
「(1+2+3+4+5)%5 = 3」という演算結果を示す。

明示的定義をした動詞は次のようにしてタシットへ変換できる。

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

他の品詞についても、それぞれに呼応して１０～１２にコロンを付して変換できる。
（ただし変換が可能な場合に限る。）

【副詞の定義の例】

```
   newton=.1 :'x. % x.D.1'^:_
   pol=._2 _1 1&p.
   pol"0 i.5
_2 _2 0 4 10
   pol newton 1
2
   pol newton 0
_1
```

```
   4!:0< 'newton'
1
```

確かに「newton」は副詞である。
「－２－ｔ＋ｔ2」という多項式の
「０,１,２,３,４」に於ける値である

多項式の根は２である(ニュートン法)

初期値を変えるともう１つの根

５．２
制御構文

【Ｊはリリース２から「$．」を廃止し、制御構文を採用した。制御構文は、明示的定義を行う際に利用できる。】

```
if. A do. B end.
```
Aが「0」でない“真”のときにＢを実行する。
```
if. A do. B1 else. B2 end.
```
Aが「0」でない“真”のときにＢ１を実行し、「0」のときにはＢ２を実行する。
```
if. A do. B1 elseif. C1 do. B2
           elseif. C2 do. B3 end.
```
Aが「0」でない(真の)ときＢ１を実行し、Aが「0」でＣ１が「0」でないときにＢ２を実行し、さらにＣ１が「0」でＣ２も「0」のときＢ３を実行する。
```
while. A do. B end.
```
Aが「0」でない(真である)限りＢを実行する。
```
whilst. A do. B end.
```
まずＢを実行し、Aが「0」でない(真である)限りＢを実行する。
```
try. B1 catch. B2 end.
```
Ｂ１を実行し、エラーがあればＢ２を実行する。
```
for. B1 do. B2 end.
```
「Ｂ１」の回数だけ「Ｂ２」を実行する。
```
select. Y
    case. 1 do. D1
    case. 2 do. D2
    case. 3 do. D3
end.
```
「Y＝１,２,３」に呼応して「Ｄ１,Ｄ２,Ｄ３」を実行する。

【さらに、次のような制御命令も用意されている】

break.	while(whilst)構文から抜け出す。
continue.	while(whilst)構文を続ける。
goto_name.	ラベル名‘name’へジャンプする。
label_name.	gotoのジャンプ先
return.	構文の流れから抜け出す。

```
   iota1=.3 :0
n=.s=.0:`$@.(]=0:)y.
for. i.<:|y. do. s=.s,n=.>:n end.
|.^:(_1=*y.)s
)
   zero1=.3 :0
if.y.=0 do.goto_0. else.goto_1. end.
   label_0. 'zero' return.
   label_1. 'nonzero'
)
```

```
   iota1&.> 5;0;_4
┌─────────┬─┬─────┐
│0 1 2 3 4│ │3 2 1 0│
└─────────┴─┴─────┘

   zero1 5
nonzero
   zero1 0
zero
```

```
   zero=.3 :0
if. y.=0 do. 'zero' else.' nonzero' end.
)
```

```
   zero 3
nonzero
   zero 0
0
```

```
   pzm=.3 :0
if. y.=0 do.":0 elseif. y.>0 do.":1
               elseif. y.<0 do.":_1 end.
)
```

```
   pzm"0(3 0 _1)
1
0
_1
```

```
   fact=.3 :0
f=.n=.1+y.
while. n>1 do. f=.f*n=.n-1 end.
)
```

```
   fact"0 i.6
1 1 2 6 24 120
   ! i.6
1 1 2 6 24 120
```

```
   box=.3 :0
b='' [ n=.>:y.
whilst. n>1 do. b=.b,<i.n=.<:n end.
|.b
)
```

```
   box 4
┌─┬─┬─┬─┐
│0│0 1│0 1 2│0 1 2 3│
└─┴─┴─┴─┘
```

「([:<＼i.)4」としても同じ結果が得られる

```
   sum=.3 :0
try. +/1+i.0>.y. catch.'miss!' end.
)
```

```
   sum 10
55
   sum '10'
miss!
```

```
   fact1=.3 :0
f=.n=.(([]>:0:)*1:>.|)y.
for.i.<:n do. f=.f*n=.<:n end.
)
```

```
   fact1"0 i.6
1 1 2 6 24 120
   ! i.6
1 1 2 6 24 120
```

```
   iota=.3 :0
s=.0:`$@.(]=0:)y.
for_j. i.|y. do.
   if.j=0 do. continue. end.
   s=.s,j
end.
|.^:(_1=*y.)s
```

```
   iota 5
0 1 2 3 4
   iota 0

   iota _6
5 4 3 2 1 0
```

「i.5」,「i.0」,「i._6」と同じ結果

```
   case=.3 :0
select. y.
   case. 1 do.i.1
   case. 2 do.i.2
   fcase.3 do.i.3
end.
)
```

```
   case&.> <"0 i.5
┌─┬─┬─┬─┬─┐
│ │0│0 1│0 1 2│ │
└─┴─┴─┴─┴─┘
```

「1,2,3」以外の引数には「空('')」を返す。

５．３　定義と引数の制限用名詞・副詞・接続詞

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
=. =:		（無し）	Copula (is) 連結詞 (名詞) X =. Y X =: Y	「=.」は局所(local)定義 「=:」は大局(global)定義 名詞に数値や文字等を挿入 明示的定義による関数を実行したときに、局所定義の場合は影響を受けないが、大局定義の場合には以前の定義が変更される。 test=. :3 (a=.+:y.),b=:-:y.) a=.b=.100 a,b 100 100 test 10 20 5 a,b 100 5 局所定義のａのみ元のまま 'a b c'=.1 2 3 ａ,ｂ,ｃに１,２,３が挿入 'X Y Z'=.i.3 2 Ｘには(0 1)、Ｙには(2 3) Ｘには(4 5)が挿入される 代名詞の衝突を避けるにはロケール(locale)を使用 square_XYZ_=.*: （ＸＹＺロケールの中の関数として利用できる）
[(]) Same 同等 （名詞）		定義内容を表示させる。] n=.i.2 3 0 1 2 3 4 5	Left/Right 右引用 左引用	[は左引数を取り出す] は右引数を取り出す 2 3 [4 5 2 3 2 3] 4 5 4 5
[. (].) Lev(Dex) 左(右)継承 （接続詞）		左（右）の関数を継承する。 sqrt=.%: [. sqr=.*: sqr 1 2 3 1 4 9 sqrt 1 2 3 1 1.41421 1.73205	（無し）	

x. y. m. n. u. v. Surrogate-arguments 代用値 (名詞)		明示的定義の際の引数 【左】【右(又は単項)】 x.　y.　汎用 m.　n.　名詞 u.　v.　動詞 [,] もｘ．，ｙ．と同一の記号として使用可能である。		

]: Identity （副詞）		右引数だけに固定する。 +: 1 2 3 4 2 4 6 8 +:]: 1 2 3 4 2 4 6 8 0 +:]: 1 2 3 4 domain error(両側演算不可)		
: Monad/Dyad 単項／二項 （接続詞）		「(単項) : (二項)」の形で定義し、左引数の有無によって単項又は二項として演算する log=.10&^. : ^. log 10 100 1 2 8 log 10 100 1.10731 2.21462	（無し）	

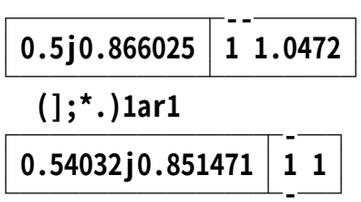
５．４　数値と文字の相互変換

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
;: Word Formation 単語生成 ;: Y		文字列をボックスで区切る。 ;:'abc XYZ' ┌─┬─┐ │abc│XYZ│ └─┴─┘ > ;:'abc XYZ' abc XYZ	（無し）	
": Default Format デフォルト 書式 ” : Y		数値で与えた右引数を文字化する。]y=.":i.2 5 0 1 2 3 4 5 6 7 8 9 $ y 2 9 文字化されている！	Format 書式 X ": Y	書式を整え文字化して表示 整数部で全体の桁数、小数部で小数以下の桁数を示す 6.2 ": 1 2 3 %3 0.33 0.67 1.00 3.2 ": 1 2 3 %3 ********* スペースがないと表示せず
". Do 実行 ". Y		文字列で与えた右引数を数値化して実行する。 a=.'3+4+5' ". a 12	Numbers X ". Y	Ｙを数値化したときに欠けた部分をＸで補う。 y=.'1 2 3','4 5',:'' 10 ". y 1 2 3 4 5 10 10 10 10

５．５　名詞の生成

演算結果表示の優先は、複素数、不動小数点、分数、拡張整数、整数、ブール数の順。

品　詞	結果と例	品　詞	結果と例
ad ar	Degreeによる複素数表示 Radianによる複素数表示 (];*.)1ad60	b （名詞）	左に進数の値右にその進数での値を置いたときの１０進数の値 16b23

（名詞）			35 （「(2*16)+3」の値） 16b1f 31 （ａ～ｚで10～35を表わす） 2b111 7 #: 7 1 1 1
e （名詞）	１０進数の桁数表示 2.3e_2 0.023	j （名詞）	複素数表示。 「3j4」は実部３虚部４の複素数
p （名詞）	「ｍｐｎ」は(ｍπ)のｎ乗 2p1 6.28319 1p2 9.8696 （「1p1」の２乗）	r （名詞）	有理数の分数表示。 2r3 0.666667 ver.3.05からは、「2r3」のように有理数の方が優先されるようになった。「2r_4」は「_1r2」
x （名詞）	「ｍｘｎ」は(ｍｅ)のｎ乗 1x1 2.71828 （オイラーの定数） 2x3 40.1711	x(末尾) （名詞）	長い実数表示を指定する。]a=.123456789x 123456789x *: a 15241578750190521 3 % a 1r41152263 ver.3.04以前は有理数表示が優先されないのでdomain error

x: Extended precision 拡張精度 （動詞）	分数表示の実数の拡張表現。 1 x: 1.2 6r5 2 x: 1.2 3r42 6 5 1 14 有理数表現の分子と分母を示す x: o.1 12852902862149r409120605684 （「π」の分数表現を与える）	a. Alphabet アルファベット	アルファベットなどの２５６のキャラクターを表示する。]s=.a.i.'aA' 97 65 ａ，Ａの位置を検出している。 (s+/i.26){a. abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
a: Ace （名詞）	空のボックスのリスト	NB.	コメント mean=.+/%# NB.mean mean 1 2 3 4 5 3 NB. 以降の行は演算に関係せず

５．６　ユーティリティ

用　法	APL	結果と例（単項）	用　法	結果と例（二項）
b. Basic Character-ristics （動詞） u b.n		「ｕb.0」は動詞(ｕ)のランクを表示する。 *: b.0 0 0 0 「(単項),(二項左),(二項右)」 *: b._1	Boolean (2.2)	

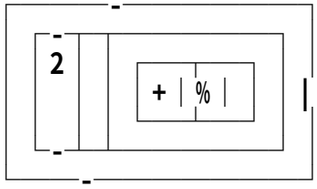
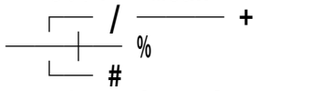
		%: （逆関数の表示） * b.1 $&一@(}.@$) （プリミティブの原始定義）		
f.		（無し）	Fix 動詞の固定 （副詞）	動詞の定義を固定し、その後の変更の影響を排除する sum=.+/ mean1=.sum % # mean=.mean1 f. mean 1 2 3 4 5 3]sum=.-/ -/ mean1 1 2 3 4 5 0.6 mean 1 2 3 4 5 3 後の変更でも「mean」は不変

!. Fit Customize フィット カストマイズ （接続詞）		許容誤差 (^!._1)〜 i.5 1 1 2 6 24 ([:*/]+_1:*i.)"0 i.5 1 1 2 6 24 (^!.1)〜 i.5 1 1 6 60 840 ([:*/]+i.)"0 i.5 1 1 6 60 840	（無し）	
:. Obverse オブバース 逆定義 （接続詞）		逆定義が正しく定義されたかどうかを確かめる。 f=.*: :. %: g=.*: :. +: f i.5 0 1 4 9 16 g i.5 0 1 4 9 16 先の動詞だけが実行される f ^:_1 f i.5 0 1 2 3 4 逆関数が正しく定義されていれば右引数の値を帰す。 g ^:_1 g i.5 0 2 8 18 32 「+: *: y」の演算結果で正しく定義されていない。	（無し）	

第６章　外部接続詞

Ｊでは、外部接続詞により、いろんなシステム関数を提供している。使いやすい名前を付してプロファイルに登録しておくと便利である。勿論、ダイレクトでも構わず、プログラム内でも使用できる。

番　号	ＡＰＬ	機能・定義の例	用　例
０ SCRIPTS	) ｉｎ	イン・スクリプトファイルの読み込み(プログラムのロード) 0!:k y 0!:2 test script	「profile」は起動時に自動的に読み込まれる。「profile.ijs」というファイルを作り、よく使われる外部接続詞を登録して置くと便利 in=.0!:2@< newton=.'c:＼jpc＼qr＼newton.js' in newton 「0!:3」は非表示で読み込み 0!:11 <'test' スクリプトをとる。
１ FILES		1!:0 y：directry 1!:1 y：read x 1!:2 y：write x 1!:3 y：append 1!:4 y：size 1!:5 y：create dir x 1!:6 y：query/set attribute x 1!:7 y：query/set permission 1!:11 y：indexed read x 1!:12 y：indexed write 1!:20 y：file numbers/mames 1!:21 y：open 1!:22 y：close 1!:30 y：locks 1!:31 y：lock 1!:32 y：unlock 1!:55 y：erase/file	1!:0<'c:＼jpc＼js＼eval.js' eval.js　1998-9-26 1!:1<'c:＼jpc＼js＼eval.js' 読み込みのみで、行を「ENTER」でたたけば動く。 'DATA'1!:2<'filename' データをセーブする。 'DATA'1!:3<'filename' データを追加する。 1!:4 '' 1156 データのサイズを表示する。
２ HOST COMMANDS		ホスト　シェル 2!:0 y：host 2!:1 y：spawn 2!:55 ''：終了	
３ STORAGE TYPES	ＮＣ	3!:0 y：データの型の判別 3!:1 y：内部表現の２進表示 3!:2 y：「3!:1 y」の逆変換 3!:3 y：内部表現の１６進数	「3!:0 y」のリターンコード 1：Boolean 2：Literal 4：Integer 8：Floating point 16：Complex 32：Boxed

			64：Extended Integer 128：Rational
４ NAME CLASS LIST		4!:0<'y'：定義内容の品詞 4!:1 y：name list 4!:3<'y'：script 4!:4<'y'：script index 4!:55<'y'：erase 4!:56 y：erase all	「4!:0 y」のリターンコード 0：Noun 1：Adverb 2：Conjunction 3：Verb 6：locale
５ REPRESENT- ATION		x 5!:0 'y'：Define 5!:1< 'y'：Atomic 5!:4< 'y'：Tree 5!:5< 'y'：Linear 5!:6< 'y'：括弧表現 (5!:1<'mean')5!:0 +/ % #]a=.o.i.2 3 0 3.14159 6.28319 9.42478 12.5664 15.708 5!:5 <'a' 3.1415926535897931*i.2 3	「5!:0 y」のリターンコード 0：Noun 2：Hook 3：Fork 4：Bonded Conjunction 5：Bident 6：Trident 7：Defined operator h=.+% 5!:1< 'h'  mean=.+/%# 5!:4 <'mean'  5!:5 <'mean' +/ % # 5!:6 <'mean' (+/) % #
６ TIME	Ｔｓ	タイムスタンプ 6!:0 'y'：現在時間	6!:1 'y'：経過時間 6!:2 'y'：実行時間
７ SPACE		使用メモリーの量 7!:0 'y'：現在使用量 7!:1 'y'：セッション使用量	7!:2 'y'：センテンス使用量 7!:3 'y'：Ｊメモリー使用量 7!:4 'y'：メモリーの開放

９ GLOBAL PARAMETERS		使用メモリーの量 9!:0 'y'：現在使用量 9!:1 'y'：Random Speed 9!:2 'y'：Default Displays 9!:3 'y'： 9!:6 'y'：ボックス枠のキャラクター 9!:7 'y'：ボックス枠の変更 9!:8 'y'：Error Message 9!:9 'y'：French Error Message 9!:14 ''：Ｊの「Version」 9!:18 ''：計算精度の変更

11 WINDOWS		11!:0 'y'：Window Driver	
13 DEBUG		13!:0 y：reset 13!:1 y：display stack 13!:2 y：stop query 13!:3 y：stop set	13!:4 y：run again 13!:5 y：run next 13!:6 y：exit & return 13!:7 y：continue 13!:16 y：「Appendix」参照
14 DATA DRIVER		14!:0 y：connect 14!:1 y：disconnect 14!:2 y：SQL 14!:3 y：Fetch 14!:4 y：Columns 14!:5 y：Column name 14!:6 y：Source name 14!:7 y：Select 14!:8 y：End 14!:9 y：Error 14!:10 y：Transaction begin 14!:11 y：Commit transaction 14!:12 y：Roll back transaction 14!:13 y：Table	
15 DYNAMIC LINK LIBRARY		15!:0 y：Call DLL function 15!:1 y：Memory read 15!:2 y：Memory write 15!:3 y：Allocate memory 15!:4 y：Free memory	
128 NUMERICAL FUNCTIONS		128!:0 y：ＱＲ分解 128!:1 y：上三角行列へ分解	

《ＡＰＰＥＮＤＩＸ　》

【Ｊの名前の由来】
《Ｑ》なんでＪと称するのか？
《Ａ》　　キーボードの右手の親指のホームポジション
　　開発者である「R.Hui」と「K.E.Iverson」のＨ.Ｉ.とＫ.の間の文字
　　大リーグ優勝を初めてカナダにもたらした「Tront BlueJays」
　いろんな憶測がなされているが、真相は定かでない。

【ＡＰＬとの相違】
《Ｑ》ＡＰＬとどこが違うの？
《Ａ》大きくは２点あって、使用キャラクターと、より関数型言語に近づけた点である。つまりＡＰＬでは、特殊文字を用いているため、通信や印刷などで難点が多かった。アスキー文字だけを使用してこの難点を解消した。またＪ言語による「Tacit Definition」は、完成度の高い関数型言語といえる。ＡＰＬでも、関数だけで表現できるタイプのものをサポートし始めたようである。

【アレイについて】
《Ｑ》アレイとは？
《Ａ》「アレイ(Array)」という言葉は“整列させる”という意味で動詞でもあるが、
　名詞としても使われ、「整列」とか「配列」といった意味である。
　アトム(Atom)：０次元の要素で、数学でいう「スカラー」に対応する。
　リスト(List)：１次元の要素で、数学でいう「ベクトル」である。
　テーブル(Table)：２次元の要素で、数学でいう「マトリクス」である。
　アレイ(Array)：アトム,リスト,テーブル等の総称で、３次元以上の要素から成る
　　一般アレイも扱うことができる。ただしＪでは、“次元”という
　　表現は用いていない。対応する言葉は“ランク”である。
　軸(Axis)　アトムは軸がなく、リストは１軸、テーブルには１,２軸がある。つまり、アレイの次元数と軸の数とは同じである。
　形(Shape)　各軸に含まれるアトムの数である。特にアトムの形は「空」になる。
　ランク(Rank)　軸の数で、アレイの次元数と同じである。
　セル(Cell)　アレイ自身も含めて、より低次のランクをもった構成要素をすべて
　　セルという。例えば（２,４)の形のアレイはランク２のセルが１つ、
　ランク１のセルが２つ、そしてランク０のセルが８つある。
　アイテム(Item)　アレイの１つランクの下がったセルをアイテムという。リストは
　　アトムで構成されているから、リストのアイテムはアトムである。
　　またテーブルのアイテムはリスト。ただしアトムのアイテムは、
それ自身のアトムがアイテムになる。
　　ランク指定のない動詞は、すべてアイテムに作動する。

【品詞について】
《Q》Ｊ言語にはどうして「品詞」があるの？
《A》人間が用いている言語には、すべて「品詞」という概念がある。コンピュータ言語　　にも「品詞」という考えを導入する方が自然だとアイバーソン等は考えた。
《Q》動詞とは？
《A》いわゆる（ＡＰＬなどでいう）関数のことである。
《Q》副詞とは？
《A》動詞をパワーアップする作用素のこと。
《Q》では接続詞は？
《A》複数の品詞を結合してより高機能な動詞や副詞を作るためのもの。Ｊでは、タシッ　　トによる定義で複数個の動詞を並べた場合、ＡＰＬなどのように右から順に作動す　　るとは限らない。三連動詞の場合のフォーク(Fork)、二連動詞のフック(Hook)とい　　った特殊な結合をする。そこで、右から順に作動させたいようなときには、接続詞　　で連結するか、または単項動詞の左にキャップ([:)をつけて合成する。

【重複定義について】
《Q》重複定義すると前に定義したもの消えてしまうの？
《A》関数型言語の重複定義への対処法には次のタイプがある。
　　重複定義は認めない。
　　重複定義では警告を出して、許容指示を与えないと書き替えない。
　　警告を出さずに前の定義を消して、新しいものに書き替えてしまう。
　Ｊは　のタイプなので注意が必要である。
《Q》　か　のほうが安全なのでは？
《A》そのためにＪでは、「ロケール」の使用を推奨している。

【ボックスについて】
《Q》ボックスとは？
《A》Ｊでは、名詞(変数)の種類として、数値と文字、それにボックスがある。
　このボックスはＪ独特のもので、馴れてしまえば非常に重宝なものである。
　新しい「Version」では「Ｌ：０」などの接続詞が導入されてボックスのまま
　演算させることができるようになっている。
《Q》ボックスを作るには？
《A》「<」,「;」,「{」などがある。
《Q》「<」や「;」の使い方は分かるのですが、「{」がよく分からない。
《A》「カタログ」と呼ばれる動詞で、「{」を両側形として使うときとは、まったく
　働きが違うので、注意する必要がある。片側形では、目録やテーブル等を作るとき
　に結構便利な機能である。
　例えば、次のように「カタログ」を使うと、５２枚のカードのセットを求めることができる。

```
  { 'CDHS' ; '23456789TJQKA'
```

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

```
   card=.3 :0
a=./:~y.?52
b=.{ 'CDHS' ; '23456789TJQKA'
c=.+/a="0 2 i.4 13
c #"1 b
)
   card 13
```

C9	CT				

D6	DT	DK			
H4	H5	H8	HJ	HQ	HA
S6	SA				

これは、１３枚カードを抽出した“ブリッジ・ハンド”である。
また次のようにすれば、「九・九の表」が得られる。

```
L=.{ n ; 'x' ;(n=.,":,.2+i.8); '='
R=.": L:0 <"0 */～ 2+i.8
L , L:0 R
```

2x2=4	3x2=6	4x2=8	5x2=10	6x2=12	7x2=14	8x2=16	9x2=18
2x3=6	3x3=9	4x3=12	5x3=15	6x3=18	7x3=21	8x3=24	9x3=27
2x4=8	3x4=12	4x4=16	5x4=20	6x4=24	7x4=28	8x4=32	9x4=36
2x5=10	3x5=15	4x5=20	5x5=25	6x5=30	7x5=35	8x5=40	9x5=45
2x6=12	3x6=18	4x6=24	5x6=30	6x6=36	7x6=42	8x6=48	9x6=54
2x7=14	3x7=21	4x7=28	5x7=35	6x7=42	7x7=49	8x7=56	9x7=63
2x8=16	3x8=24	4x8=32	5x8=40	6x8=48	7x8=56	8x8=64	9x8=72
2x9=18	3x9=27	4x9=36	5x9=45	6x9=54	7x9=63	8x9=72	9x9=81

【取りと落しについて】
《Q》取りと落しの機能がいろいろあってヤヤコシソーなのですが？
《A》それでは、取りと落しを対比させて説明してみよう。

```
      《先頭取り(Head)》                    《先頭落し(Behead)》
   {. 1 2 3 4                    }. 1 2 3 4
1                                2 3 4
   【右引数の先頭の要素(アイテム)を取る】          【引数の先頭の要素を落す】
   {. i.3 4                         {. i.3 4
0 1 2 3                          4 5  6  7
                                 8 9 10 11
              【先頭行(アイテム)の取り】          【先頭行(アイテム)の落し】
      《取り(Take)》                        《落し(Drop)》
   2 {. 1 2 3 4                     2 }. 1 2 3 4
1 2                              3 4
           【先頭からＸ個の要素を取る】          【先頭からＸ個の要素を落す】
   _2 {. 1 2 3 4                    _2 }. 1 2 3 4
3 4                              1 2
   【負なら末尾から(－Ｘ)個の要素を取る】        【末尾から(－Ｘ)個の要素を落す】
   2 3 {. i.4 5                     1 2 }. i.3 4
0 1 2                             6  7
5 6 7                            10 11
             【２行３列目までの要素】            【１行２列目までを落す】
   2 2 4 {. i.3 4 5                 1 2 3 }. i.3 4 5
 0  1  2  3                       33 34
 5  6  7  8                       38 39

20 21 22 23                      53 54
```

```
25 26 27 28                    |58 59
          【２枚の２行４列の要素の取り】          【１枚を除き２行３列を落す】

      《末尾取り(Tail)》                    《末尾落し(Curtail)
   {: 1 2 3 4                  |   }: 1 2 3 4
4                              |1 2 3
            【右引数の最後の要素を取る】          【右引数の最後の要素を落す】
   {: i.3 4                    |   }: i.3 4
8 9 10 11                      |0 1 2 3
                               |4 5 6 7
              【最終行(アイテム)の取り】          【最終行(アイテム)の落し】

《Q》指標(インデックス)で指定する取りや落しは？
《A》選択(From)と呼ばれている「 {」とボックス「<」を用いればよいのだが、これは
     結構複雑である。

   ]A=.i.4 5                   |   ]B=.i.2 3 4
 0  1  2  3  4                 | 0  1  2  3
 5  6  7  8  9                 | 4  5  6  7
10 11 12 13 14                 | 8  9 10 11
15 16 17 18 19                 |
                               |12 13 14 15
                               |16 17 18 19
                               |20 21 22 23

   0 { A                       |   0 { B
0 1 2 3 4                      |0 1  2  3
                               |4 5  6  7
                               |8 9 10 11
                        【０行の取り出し】          【０枚目の取り出し】
   0 1 { A                     |   2 3 {"1 A
0 1 2 3 4                      | 2  3
5 6 7 8 9                      | 7  8
                               |12 13
                               |17 18
                    【０,１行の取り出し】    【１軸を指定すると２,３列の取り出し】  0 {"1 B
                     |   0 {"2 B
 0  4  8                       | 0  1  2  3
12 16 20                       |12 13 14 15
       【１軸を指定すると０列を取り出す】      【２軸指定の場合は０行を取り出す】
   (<2 3){ A                   |   (<0 1){ B
13                             |4 5 6 7
       【２行３列の要素(ボックスで指定)】          【Ｂの０枚目の１行目の要素)】
   (<0 1 2){ B                 |   (<0;1;2){ B
6                              |6
            【いずれもＢの(０,１,２)の要素で、ＡＰＬではＢ[１;２;３]に対応する】
   (0 1; 1 2) { B              |   ((<0 1)&{,:(<1 2)&{) B
 4  5  6  7                    | 4  5  6  7
20 21 22 23                    |20 21 22 23
          【０枚目の１行と１枚目の２行】
   (0 1 2; 1 2 3) { B          |   ((<0 1 2)&{,(<1 2 3)&{) B
6 23                           |6 23
                               |  【Ｂの(０,１,２)と(１,２,３)の要素を取り出す】
   (<<1){ 1 2 3 4 5            |   (<<1){ A
2                              |5 6 7 8 9
              【指標１の要素を取り出す】          【指標１の行(アイテム)を取る】
   (<2 3;3 4){ A               |   (2 2$2 3;2 4;3 3;3 4){ A
```

```
13 14                              | 13 14
18 19                              | 18 19
```
【２,３行と３,４列のブロックを取り出す(２重ボックスで指定)】

```
   (<0 1; 1 2) { B                 |    (2 2$0 1;0 2;1 1;1 2){ B
 4  5  6  7                       |  4  5  6  7
 8  9 10 11                       |  8  9 10 11
                                   |
16 17 18 19                        | 16 17 18 19
20 21 22 23                        | 20 21 22 23
```
【０,１面と１,２行のブロックを取り出す】

《Q》インデックスを指定して落すには？
《A》「３重ボックス」を使えばよい。

```
   (<<<1){ 1 2 3 4 5               |    $ (<<<1){ 1 2 3 4 5
1 3 4 5                            | 4
```
【指標１の要素を落して、ランクが１つ落ちたリストになる】

```
   (<<<1){ A                       |    $ A
 0  1  2  3  4                     | 4 5
10 11 12 13 14                     |    $ (<<<1){ A
15 16 17 18 19                     | 3 5
```
【結果はＡの１行目を除いたもので、０軸のランクが１つ落ちる】

```
   (<<<1){ B                       |    $ B
0 1  2  3                          | 2 3 4
4 5  6  7                          |    $ (<<<1){ B
8 9 10 11                          | 1 3 4
```
【結果はＢの１面目を除いたもので、やはり０軸のランクが１つ落ちる】

```
   (<<<1){"1 A                     |    $ (<<<1){"1 A
 0  2  3  4                        | 4 4
 5  7  8  9                        |
10 12 13 14                        |
15 17 18 19                        |
```
【ランクを指定すると結果はＡの１列目を除いたもので、１軸のランクが１つ落ちる】

```
   $(<<<2){"1 B                    |    $ (<<<2){"2 B
2 3 3                              | 2 2 4
```
【Ｂの２列目(１軸)が落される】　　【Ｂの２行目(２軸)を落す】

【カレンダー】
《Q》「軸」と「アイテム」といった概念がよく分からない。
《A》では、カレンダーの例で説明してみよう。
　　「CALENDAR97」には、1997年のカレンダーが挿入されている。

```
  4 3$<"2 CALENDAR97
```

0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 0 0 0 0 0
0 0 1 2 3 4 5 6 7 8 9 10 11 12	0 0 0 0 1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10 11 12 13 14

```
 13 14 15 16 17 18 19 | 11 12 13 14 15 16 17 | 15 16 17 18 19 20 21
 20 21 22 23 24 25 26 | 18 19 20 21 22 23 24 | 22 23 24 25 26 27 28
 27 28 29 30  0  0  0 | 25 26 27 28 29 30 31 | 29 30  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  0  1  2  3  4  5 |     0  0  0  0  1  2 |  0  1  2  3  4  5  6
  6  7  8  9 10 11 12 |  3  4  5  6  7  8  9 |  7  8  9 10 11 12 13
 13 14 15 16 17 18 19 | 10 11 12 13 14 15 16 | 14 15 16 17 18 19 20
 20 21 22 23 24 25 26 | 17 18 19 20 21 22 23 | 21 22 23 24 25 26 27
 27 28 29 30 31  0  0 | 24 25 26 27 28 29 30 | 28 29 30  0  0  0  0
  0  0  0  0  0  0  0 | 31  0  0  0  0  0  0 |  0  0  0  0  0  0  0
---------------------+----------------------+---------------------
  0  0  0  1  2  3  4 |  0  0  0  0  0  0  1 |  0  1  2  3  4  5  6
  5  6  7  8  9 10 11 |  2  3  4  5  6  7  8 |  7  8  9 10 11 12 13
 12 13 14 15 16 17 18 |  9 10 11 12 13 14 15 | 14 15 16 17 18 19 20
 19 20 21 22 23 24 25 | 16 17 18 19 20 21 22 | 21 22 23 24 25 26 27
 26 27 28 29 30 31  0 | 23 24 25 26 27 28 29 | 28 29 30 31  0  0  0
  0  0  0  0  0  0  0 | 30  0  0  0  0  0  0 |  0  0  0  0  0  0  0
```

```
   ($ ; $&$) CALENDAR97
-----
12 6 7 | 3
-----
```

だから、これは「ランク３」の「アレイ」で、(12,6,7)という「形」をしている。

```
   {. CALENDAR97                |    {: CALENDAR97
 0  0  0  1  2  3  4           |  0  1  2  3  4  5  6
 5  6  7  8  9 10 11           |  7  8  9 10 11 12 13
12 13 14 15 16 17 18           | 14 15 16 17 18 19 20
19 20 21 22 23 24 25           | 21 22 23 24 25 26 27
26 27 28 29 30 31  0           | 28 29 30 31  0  0  0
 0  0  0  0  0  0  0           |  0  0  0  0  0  0  0
```

【最初の月(アイテム)は１月】　　【最後の月(１２月)】

また８月のカレンダーは８番目の月で、インデックス７のアイテムである。

```
   ] AUG=.7{ CALENDAR97         |    ] WEEK4=.3{ AUG
    0  0  0  0  1  2           | 17 18 19 20 21 22 23
 3  4  5  6  7  8  9           |    3 { WEEK4
10 11 12 13 14 15 16           | 20
17 18 19 20 21 22 23           |    (<7 3 3){ CALENDAR97
24 25 26 27 28 29 30           | 20
31  0  0  0  0  0  0           |
```

【８月の第４週の水曜日の日付は「２０」である】

「CALENDAR97」というランク３のアレイでは、月(面)方向を０軸、週(列)方向を１軸、曜日(行)方向を２軸という。

「CALENDAR97」のアイテムは、JAN,FEB,･･･････,AUG,･･･････,DEC

【ランクは３で、１２個のアイテム(テーブル)をもっている】

「AUG」のアイテムは、第１週，第２週，･･･････，第６週

【ランクは２で、６個のアイテム(リスト)がある】

「WEAK4」のアイテムは、日曜，月曜，･･･････，土曜

【ランクは１で、７個のアイテム(アトム)がある】

《Q》ランク指定の「取り({)」がヤヤコシイ。

《A》これも、カレンダーの例で確かめてみよう。

```
   ] WED=.3{"1 CALENDAR97       |    ] WEEK_4=.3{"2 CALENDAR97
1  8 15 22 29 0                 | 19 20 21 22 23 24 25
```

```
0   5  12  19  26  0                 | 16  17  18  19  20  21  22
0   5  12  19  26  0                 | 16  17  18  19  20  21  22
2   9  16  23  30  0                 | 20  21  22  23  24  25  26
0   7  14  21  28  0                 | 18  19  20  21  22  23  24
4  11  18  25   0  0                 | 22  23  24  25  26  27  28
2   9  16  23  30  0                 | 20  21  22  23  24  25  26
0   6  13  20  27  0                 | 17  18  19  20  21  22  23
3  10  17  24   0  0                 | 21  22  23  24  25  26  27
1   8  15  22  29  0                 | 19  20  21  22  23  24  25
0   5  12  19  26  0                 | 16  17  18  19  20  21  22
3  10  17  24  31  0                 | 21  22  23  24  25  26  27
        【各月の水曜日(第１軸の第３列)】          【各月の第４週(第２軸の第３行)】
```

日付をサーチする関数を次のように与える。

```
  what_day=.<@[{]
   7 3 3 what_day CALENDAR97
20
   7 3 4 what_day CALENDAR97
21
   8 3 3 what_day CALENDAR97
24
```

【ボックスによる回帰直線の同時計算】
《Q》ボックスのままで計算を行なうには？
《A》回帰直線を求める例で説明してみよう。

```
  X;Y
┌─────┬─────┐
│1 2  │2 3  │
│2 3  │1 2  │
│3 4  │3 4  │
└─────┴─────┘
```

```
  {@|:@] X
┌─────┬─────┐
│1 2 3│2 3 4│
└─────┴─────┘
```

```
  (1:,.L:0{@|:])X
┌─────┬─────┐
│1 1  │1 2  │
│1 2  │1 3  │
│1 3  │1 4  │
└─────┴─────┘
```

```
  Y([%.L:0 1:,.L:0{@|:])X
┌─────┬─────┐
│  1  2 │0.5 1.5│
│0.5 0.5│0.5 0.5│
└─────┴─────┘
```

【ボックスの中で１を加えている】　　【４つの回帰直線が同時に算出される】

```
   |:> Y([%.L:0 1:,.L:0{@|:])X
  1 0.5
0.5 0.5

  2 0.5
1.5 0.5
```

【「Tacit」と「Explicit」の結合】

《Q》Ｊでは関数を定義するのに２種類の方法があるが、どちらが良いか？
《A》個人的な好みもあるので、一概にどちらが良いとは決められない。ただ「Tacit」のほうがＪ独特のものだから、「Explicit」だけ使うのではチト芸がない。一般に平均や移動平均などの例のように、少ないキャラクターでプログラムできるような場合は、「Tacit」のほうが良いと思う。しかし、ループや分岐を使って組む必要があるケースでは、「Explicit」で定義するほうが容易だろう。

《Q》複数個の関数を結合して別の新しい関数が定義できるが、「Tacit」と「Explicit」とを混ぜこぜにして結合できるのか？
《A》一般に、混ぜこぜにして構わない。また「Explicit」の定義の際に、中で「Tacit」の関数を使うこともできる。シンプルで、分かりやすいプログラムがベターだ。

```
   sub1=.+:
   sub2=.3 :'3*y.'
   sub3=.+:^:2
   main=.(sub1,sub2,sub3)"0
   main 3 4
6  9 12
8 12 16
```

また、次のようにまとめて定義することもできる。

```
   (3 :'2 3 4*y.'"0)3 4
6  9 12
8 12 16
```

```
   (2 3 4"_*])"0(3 4)
6  9 12
8 12 16
```

```
   (2 3 4&*"0)3 4
6  9 12
8 12 16
```

```
   (*&2 3 4"0)3 4
6  9 12
8 12 16
```

また、次のように“両側形”として定義してもよい。

```
   2 3 4 (*"1 0)3 4                 |    2 3 4 (*/～)3 4
6  9 12                             | 6  9 12
8 12 16                             | 8 12 16

   2(]*/[+[:i.3:)3 4                |    2 3(]*/({.+[:i.{:)@[)3 4
6  9 12                             | 6  9 12
8 12 16                             | 8 12 16
```

さらに、次のように「Explicit」で定義しても最後の結果と同じである。

```
   4 :'y.*/({.x.)+i.{:x.'
   4 :'y.*/({.+[:i.{:)x.'
```

【トレースモードについて】

《Q》
《A》
```
   trace=.13!:16
```
ここで、「trace」は接続詞である。
```
   mean=.+/ % #
   mave=.+/＼ % [
   _ _ trace _
```
と入力すれば、「Trace Mode」になる。
```
   mean 1 2 3 4 5
┌──────────────────────────────── 0 monad ────────────┐
mean
1 2 3 4 5
└─────────────────────────────────────────────────────┘
 +/ % #
 1 2 3 4 5
 3
 ============================
3
============================
```
といった表示が得られる。

```
   3 mave 1 2 3 4 5
┌──────────────────────────────── 2 dyad ─────────────┐
3
mave
1 2 3 4 5
└─────────────────────────────────────────────────────┘
 3
 +/＼ % [
 1 2 3 4 5
 2 3 4
 ============================
2 3 4
============================
2 3 4

   _ _ trace 0
```
とすれば、「Trace Mode」から抜ける。

【２バイト文字の処理】

《Q》日本語の漢字などは扱えるの？
《A》まず、名前のリストを名詞として入力する。
```
   name1=. 0 :0
北野武
志村健一
鈴木義朗
竹内一郎
中村隆一
)
```
このように入力したデータは、見かけ上テーブルのようになっているが、実際はラインフィールド(ＬＦ)が挿入されたリストである。そこで
```
   name2=.><;._2 name1
```
のように定義し直すと
```
   $ name2
```

5 8
のようにテーブルになる。
　ところで日本語は２バイト文字なので、日本語の処理を行なうためには、２バイトずつ囲んで処理しなければならない。

```
   ]name3=.(box=._2&(<＼"1))name2
```

北	野	武	
志	村	健	一
鈴	木	義	朗
竹	内	一	郎
中	村	隆	一

【各文字をボックスで囲む】

```
   ;"1@|. name3
```

中村隆一
竹内一郎
鈴木義朗
志村健一
北野武

【名前の順番を逆転させる】

```
   select=.[:;"1([:+/@〜."1=L:0)#]
   '一' select name3
```

志村健一
竹内一郎
中村隆一

【'一'という字を含む名前を選択する】

《索　引》

【H】

Halve(２分の１) -: 1.1
Head(先頭) {. 3.4

【I】

Identity(単項への限定)]: 5.3
Imaginary(虚数生成) j. 1.4
Increment(１増) >: 1.1
Indeterminate(不定) _. 1.2
Index of(インデックス) i. 2.1
Infinity(無限大) _: 1.2
Infix(中から) ╲ 3.6
Insert(挿入) / 3.6
Integer(整数生成) i. 3.1
Is(Local) =. 5.3
Is(Global) =: 5.3
Item Amend(アイテム修正) } 3.4
Itemize(アイテム化) ,: 3.1

【L】

Laminate(層連結) ,: 3.1
Larger-of(最大値) >. 1.1
Larger-or-Equal(小さくない) >: 2.1
Larger-Than(大きい) > 2.1
LCD(最小公倍数) *. 1.7
Left(左引用) [5.3
Left Argument x. 5.1
Length/Angle(大きさ/偏角) *. 1.4
Lesser-of(最小値) <. 1.1
Lesser-or-Equal(大きくない) <: 2.1
Lesser-Than(小さい) < 2.1
Lev(左継承) [. 5.3
Level(ボックスのレベル) L. 3.7
Level(指定レベルで演算) L: 3.7
Link(結合) ; 3.7
Logarithm(対数) ^. 1.1

【M】

Magnitude(絶対値) | 1.1
Map {:: 3.7
Match(一致) -: 2.1
Matrix Divide(行列除算) %. 1.5
Matrix Inverse(逆行列) %. 1.5
Member-In e. 2.1
Member-of-Intreval E. 2.1
Minus(-
Monad(単項) : 5.3

【N】

Natural log(自然対数) ^. 1.1
Negate(逆符号化) - 1.1
Negative-Sign(マイナス記号) _ 1.2
Not(否定) -. 2.2
Not-And(否定論理積) *: 2.2
Note Bote(注釈) NB. 5.5
Nub(重複排除) ～. 2.1
Nubsieve(重複指示) ～: 2.1

【O】

Oblieque(斜め) /. 3.8
Obverse(逆定義) :. 5.6
Odd(奇) .: 1.7
Open(オープン) > 3.7
Or(論理和) +. 2.2
Outfix(外から) ╲. 3.6
Out-of(２項係数) ! 1.8

【P】

Passive(交換) ～ 4.2
Permute(置換) C. 1.8
Pi Times(円周倍率) o. 1.3
Plus(加算) + 1.1
Polar(極座標複素数) r. 1.4
Polynomial(多項式) p. 1.7
Power(累乗) ^ 1.1
Power(反復) ^: 4.4
Prefix(前から) ╱ 3.6
Prime(素数) p: 1.6
Prime Factor(素因数分解) q: 1.6
Prime Exponent(素因数要素) q: 1.6

【R】

Rank(ランク) " 3.2
Ravel(リスト化) , 3.1
Ravel Items(テーブル化) ,. 3.1
Raze(ほぐし) ; 3.6
Raze In(部分所属) e. 2.1
Real/Imazinary(実部/虚部) +. 1.4
Reciprocal(逆数) % 1.1
Reflexive(両側化) ～ 4.2
Residue(剰余) | 1.1
Reverse(逆順) |. 3.5

Right(右引用)] 5.3

Right Augument y. 5.1
Roll(重複乱数) ? 1.6
Roll(シード固定の重複乱数) ?. 1.6
Root(平方根) %: 1.1
Rotate(回転) |. 3.5

【S】

Same Left(左引用) [5.3
Same Right(右引用)] 5.3
Secant Slope(平均変化率) D: 1.7
Self-Classify(自己分類) = 2.1
Self-Reference(自己参照) $: 4.4
Shape(変形) $ 3.1
Shape of(形) $ 3.1
Signum(符号/単位円への写影) * 1.1
Sort doun(降順ソート) ╲: 3.5
Sort up(昇順ソート) /: 3.5
Square(平方) *: 1.1
Square Root(平方根) %: 1.1
Stitch(縦連結) ,. 3.1
Suffix(外から) ╲. 3.6

【T】

Tail(末尾) {: 3.3
Talley(アイテム数) # 3.1
Take(取り) {. 3.3
Taylor Coefficient t. 1.7
Taylor Function(テイラー展開) T. 1.7
Tie(ジェランド生成接続詞) ` 4.5
Times(乗算) * 1.1
Transpose(転置) |: 3.5

【U】

Under(アンダー) &. 4.1

【W】

Weighted Taylor Coefficient t: 1.7
Word Formation(単語生成) ;: 5.4

【あ行】

アイテム化	,:	3.1
アイテム数	#	3.1
アイテム連結	,	3.1
アジェンダ	@.	4.4
アット	@:	4.1
アトップ	@	4.1
アポーズ	&:	4.1
アルファベット	a.	5.5
アンダー	&.	4.1
１増	>:	1.1
１減	<:	1.1
一致	-:	2.1
一般外積	/	1.5
一般内積	/ .	1.1
インデックス	i.	2.1
後から	╱.	3.6
ｎ進化	#:	2.3
円関数	o.	1.3
円周倍率	o.	1.3
オープン	>	3.7
大きい	>	2.1
大きいか等しい(小さくない)	>:	2.1
落し	}.	3.3

【か行】

階乗	!	1.7
回転	|.	2.1
拡張精度	x:	5.5
加算	+	1.1
形	$	3.1
カタログ	{	3.3
キー	/.	3.8
奇数接続	.:	1.9
キャップ	[:	4.2
結合	;	3.7
逆行列	%.	1.5
逆数	%	1.1
逆順	|.	3.5
逆符号	-	1.1
逆定義(オブバース)	:.	5.6
ＱＲ分解	128!:	1.5
行列除算	%.	1.5
共役複素数	+	1.1
局所定義	=.	5.3
極座標	*.	1.4
極座標表示	r.	1.4
虚数生成	j.	1.4
偶数接続	..	1.9
区切り(カット)	;.	3.8
グループ所属	E.	2.1
結合	;	3.7
減算	-	1.1
交換	～	4.2
降順インデックス	╱:	3.6
降順ソート	╱:	3.6
コメント(注釈)	NB.	5.5
コピー(複写)	#	3.3
コンポーズ	&	4.1

【さ行】

最小公倍数(LCM)	*.	1.7
最小値	<.	1.1
最大公約数(GCD)	+.	1.7
最大値	>.	1.1
自己分類	=	2.1
辞書式順序	A.	1.8
指数	^	1.1
自然対数	^.	1.1
実行(数値化)	".	5.4
修正(アメンド)	}	3.4
１０進数	#.	2.3
１０進数２進化	#:	2.3
乗算	*	1.1
除算	%	1.1
書式(文字化)	":	5.4
昇順インデックス	/:	3.5
昇順ソート	/:	3.5
所属	e.	2.1
整数生成	i.	3.1
絶対値	|	1.1
選択	{	3.3
先頭取り	{.	3.3
先頭落し	}.	3.3
挿入	/	3.6
層連結	,:	3.1
素数	p:	1.6
素因数分解	q:	1.6
素数の位置	q:	1.6
外から	╲.	3.6

【た行】

対数	^.	1.1
縦連結	,.	3.1

多項式

p.	1.7	微分(ランク固定せず)
単位複素数	r.	1.4
単語生成	;:	5.4
大局定義	=:	5.3
小さい	<	2.1
小さいか等しい(大きくない)	<:	2.1
置換	C.	1.8
注釈(コメント)	NB.	5.5
重複指示	〜:	2.1
重複排除	〜.	2.1
テーブル化	,.	3.1
テーラー展開	T.	1.7
テーラー展開の係数	t.	1.7
テーラー展開(重みづけ)	t:	1.7
デックス	].	5.3
デフォルト書式	":	5.4
転置	|:	3.5
天井値	>.	1.1
同等	][	5.3
動名詞起動	`:	4.4
動名詞作成	`:	4.4
取り	{.	3.3

【な行】

中から	╲	3.6
長さ／偏角	*.	1.4
斜め	/.	3.8
２項係数	!	1.8
２乗	*:	1.1
２乗根	%:	1.1
２倍	+:	1.1
２分の１	-:	1.1
２進数１０進化	#.	2.3

【は行】

パターンの所属	E.	2.1
反復	^:	4.4
左引用	[	5.3
左継承	[.	5.3
左引数	x.	5.5
非重複乱数	?	1.6
非重複乱数(シード固定)	?.	1.6
否定(補数)	-.	2.2
否定論理積	*:	2.2
否定論理和	+:	2.2
等しい	=	2.1
微分(ランクが０に指定)	d.	1.7

D.	1.7	
フィット	!.	5.6
フォーク		4.3
フック		4.3
複写(コピー)	#	3.3
複素数生成	j.	1.4
符号(単位円への写影)	*	1.4
不定	_.	1.2
不等	〜:	2.1
ブーリアン	b.	2.2
部分所属	e.	2.1
平均変化率	D:	1.7
平方(２乗)	*:	1.1
平方根(２乗根)	%:	1.1
変形	$	3.1
ほぐし	;	3.7
ボックス	<;	3.7
ボンド	&	4.1

【ま行】

マイナス符号	-	1.2
前から	╲	3.6
末尾取り	{:	3.3
末尾落し	}:	3.3
右引用	]	5.3
右継承	].	5.3
右引数	y.	5.3
無限大(名詞)	_	1.2
無限大(動詞)	_:	1.2
文字化	":	5.4

【や行】

床値	<.	1.1
呼出し	〜	4.2

【ら行】

ランク(階数)	"	3.2
乱数	?	1.6
乱数(シード固定)	?.	1.6
リスト化	,	3.1
両側化	〜	4.2
累乗	^	1.1
累乗根	%:	1.1
論理積	*.	2.2
論理和	+.	2.1

《プリミティブの機能一覧》

記号と読み		説明(英語)	説明(日本語)	品詞	章節
= イコール	= Y	Self-classify	自己分類を行なう	動詞	2.1
	X = Y	Equal	左右が等しければ１(論理演算)	動詞	2.1
	X =. Y	Is(Local)	局所定義	接続詞	5.3
	X =: Y	Is(Gloval)	大局定義	接続詞	5.3
< より小	< Y	Box	ボックスで囲む	動詞	3.7
	X < Y	Lesser than	ＸがＹより小なら１(論理演算)	動詞	2.1
	<. Y	Flooroval)	切り捨てた整数値	動詞	1.1
	X <. Y	Less of	小さいほうの値を与える	動詞	1.1
	<: Y	Decrement	数値から１を引く	動詞	1.1
	X <: Y	Less or Equal	小さいか等しい(論理演算)	動詞	2.1
> より大	< Y	Open	ボックスを開く	動詞	3.7
	X < Y	Larger than	ＸがＹより大なら１(論理演算)	動詞	2.1
	<. Y	Ceiling	切り捨てた整数値	動詞	1.1
	X <. Y	Large of	大きいほうの値を与える	動詞	1.1
	<: Y	Increment	数値に１を加える	動詞	1.1
	X <: Y	Large or Equal	大きいか等しい(論理演算)	動詞	2.1
_ アンダーバー	_ Y	Negative Sign	マイナス符号	動詞	1.2
	_	Infinity	無限大	副詞	1.2
	_.	Indefinite	不定	副詞	1.2
	_:	Infinity	無限大	動詞	1.2
+ プラス	+ Y	Conjugate	共役複素数	動詞	1.1
	X + Y	Plus	加算	動詞	1.1
	+. Y	Real/Imaginary	複素数の実部と虚部を与える	動詞	1.4
	X +. Y	GCD	最大公約数	動詞	1.4
	X +. Y	Or	論理和(論理演算)	動詞	2.2
	+: Y	Double	２倍にする	動詞	1.1
	X +: Y	Not-Or	否定論理和(論理演算)	動詞	2.2
- マイナス	- Y	Negative	逆符号	動詞	1.1
	X - Y	Minus	減算	動詞	1.1
	-. Y	Not	論理否定	動詞	2.2
	X -. Y	Less	Ｙに含まれないＸの要素を出力	動詞	3.3
	-: Y	Halve	半分にする	動詞	1.1
	X -: Y	Match	一致すれば１(論理演算)	動詞	2.2
* スター	* Y	Signum	符号複素数	動詞	1.4
	X * Y	Multiply	乗算	動詞	1.1
	*. Y	Length/Angle	複素数の極座標表示を与える	動詞	1.4
	X *. Y	LCM	最小公倍数	動詞	1.4
	X *. Y	And	論理積(論理演算)	動詞	2.2
	*: Y	Square	平方する	動詞	1.1
	X *: Y	Not-And	否定論理積(論理演算)	動詞	2.2
%	% Y	Reciprocal	逆数	動詞	1.1

パーセント	X % Y	Divide	除算	動詞	1.1
	%. Y	Matrix-Inverse	(一般化)逆行列を与える	動詞	1.5
	X %. Y	Matrix-Devide	行列の除算	動詞	1.5
	%: Y	Square-Root	平方根	動詞	1.1
	X %: Y	Root	累乗根	動詞	1.1
^ ハット	^ Y	Exponential	指数	動詞	1.1
	X ^ Y	Power	Ｘの数値のＹ乗	動詞	1.1
	^. Y	Natural-Log	自然対数	動詞	1.1
	X ^. Y	Logarithm	Ｘを底とする対数値	動詞	1.1
	^: Y	Power	反復計算	接続詞	4.4
	X ^: Y	Power	否定論理積(論理演算)	接続詞	4.4
$ ドル	$ Y	Shape of	Ｙの形を与える	動詞	3.1
	X $ Y	Shape	Ｘで指定した形に	動詞	3.1
	X $: Y	Self-Reference	再帰の繰返し演算	動詞	4.4
～ チルド	～ Y	Reflex	両側化	動詞	1.1
	X ～ Y	Pass Evoke	ＸとＹの交換	動詞	1.1
	～. Y	Nub	重複した要素を排除する	動詞	1.1
	～: Y	Nub Sieve	重複した要素に０を与える	接続詞	4.4
	X ～: Y	Not-Equal	等しくない(論理演算)	接続詞	4.4
縦棒	| Y	Magnitude	(実数や複素数の)絶対値	動詞	1.1
	X | Y	Residue	整数の除算の剰余	動詞	1.1
	|. Y	Reverse	アレイ(ベクトルや行列)の逆順	動詞	3.5
	X |. Y	Rotate	アレイ(ベクトルや行列)の回転	動詞	3.5
	|: Y	Transpose	アレイ(ベクトルや行列)の転置	動詞	3.5
	X |: Y	Transpose	Ｘで指定したセルの転置	動詞	3.5
. ピリオド	-/ .*Y	Determinant	一般逆行列	接続詞	1.5
	+/ .*Y	Dot-Product	一般内積	接続詞	1.5
	X .. Y	Even	偶数部分の取り出し	接続詞	1.9
	X .: Y	Odd	奇数部分の取り出し	接続詞	1.9
: コロン	m : 0	Explicit Def	明示的定義	接続詞	5.2
	u :. v	Obverse	逆定義	接続詞	5.6
	u :: v	Adverse	エラーが無ければｕ、あればｖ	接続詞	4.4

, コンマ	, Y	Ravel	リスト化	動詞	3.1
	X , Y	Append	アイテム同士の結合	動詞	3.1
	,. Y	Ravel Item	テーブル化	動詞	3.1
	X ,. Y	Stitch	高いランクの方向に結合	動詞	3.1
	,: Y	Itemize	アレイのランクを１つ増加	動詞	3.1
	X ,: Y	Laminate	高いランクの形に合わせて結合	動詞	3.1
; セミコロン	; Y	Raze	リストにほぐす	動詞	3.7
	X ; Y	Link	ボックスで囲んで接続	動詞	3.7
	;. Y	Cut	文字列やブロック等を区切る	接続詞	3.8
	X ;. Y	Cut	Ｘに応じて文字列等を区切る	接続詞	3.8
	;: Y	Word Formation	文字列をボックスで区切る	動詞	5.4
# シャープ	# Y	Tally	アレイのアイテム数	動詞	3.1
	X # Y	Copy	Ｘで指定した個数をコピーする	動詞	3.3
	#. Y	Base2	２進数の１０進数化	動詞	2.3
	X ;. Y	Base	Ｘで指定した底で１０進数値	動詞	2.3

	#: Y	Antibase2	１０進数の２進数化	動詞	2.3
	X #. Y	Antibase	１０進数のＸ進数化	動詞	2.3
!	! Y	Factorial	階乗の値を求める	動詞	1.7
感嘆符	X ! Y	Out-of	２項係数を求める	動詞	1.7
	!. Y	Fit/Customize	^ と数値を接続した階乗型関数	接続詞	5.6
	X !: Y	Foreign	外部接続詞	動詞	別掲
/	u/ Y	Insert	演算子をアイテム間に挿入する	副詞	3.6
スラッシュ	X u/ Y	Table(Insert)	左右のクロス演算(一般外積)	副詞	1.5
	/. Y	Oblique	対角線上に演算子を挿入する	副詞	3.8
	X /. Y	Key(Append)	ＹのＸに等しい部分に作動する	副詞	3.8
	/: Y	Grade up	アレイの昇順のインデクス	副詞	3.5
	X /: Y	Sort	昇順ソート	副詞	3.5
\	u\ Y	Prefix	逐次前から演算を行なう	副詞	3.6
逆スラッシュ	X u\ Y	Infix(Train)	Ｘの個数を逐次取り出して演算	副詞	3.6
	\. Y	Suffix	逐次後から演算を行なう	副詞	3.6
	X \. Y	Outfix	Ｘの個数を逐次落して演算する	副詞	3.6
	\: Y	Grade down	アレイの降順のインデクス	副詞	3.5
	X \: Y	Sort	降順ソート	副詞	3.5
[	[Y	Same/Left	左の要素を取り出す	動詞	5.3
左大カッコ	X [Y	Same/Left	Ｘだけ取り出す	動詞	5.3
	[. Y	Lev	左の関数を継承する	接続詞	5.3
	[: Y	Cap	フォークの機能を止める	動詞	4.2
	X [: Y	Cap	意図しない演算にはエラー	動詞	4.2

]	] Y	Same/Right	右の要素を取り出す	動詞	5.3
右大カッコ	X] Y	Same/Right	Ｙだけ取り出す	動詞	5.3
	]. Y	Dex	Ｙだけ取り出す	接続詞	5.3
	]: Y	Identity		副詞	5.3
{	{ Y	Catalogue	全ての組合せ	動詞	3.3
左中カッコ	X { Y	From	Ｘで指定した指標の要素の取り	動詞	3.4
	{. Y	Head	先頭の要素の取り	接続詞	3.4
	X {. Y	Take	指定個数の要素の取り出し	動詞	3.4
	{: Y	Tail	末尾の要素の取り	動詞	3.3
	X {::Y	Map	意図しない演算にはエラー	動詞	3.7
}	} Y	Item Amend		動詞	5.3
右中カッコ	X } Y	Amend	Ｘで指定した指標の要素の修正	動詞	5.3
	}. Y	Behead	先頭の要素の取り落し	接続詞	5.3
	X }. Y	Drop	指定個数の要素の取り落し	動詞	3.4
	}: Y	Identity	末尾の要素の落し	副詞	5.3
"	"n Y	Rank	ランクを指定する	接続詞	3.2
ダブルクォート	X "n Y	Rank	ランクを指定する	接続詞	3.2
	". Y	Do	数値化して実行する	動詞	5.4
	X ". Y	Number	数値化	動詞	5.4
	": Y	Format	文字化	動詞	5.4
	X ": Y	Format	Ｘで指定した書式で文字化する	動詞	5.4
`	` Y	Tie(Gerund)	条件式	接続詞	4.4
逆クォート	X `: Y	Evoke Gerund	動名詞起動	接続詞	4.4

@ アット	u @ v u @. v u @: v	Atop Agenda At	動詞の接続(ボンド) 動名詞の条件式の接続(ｱｼﾞｪﾝﾀﾞ) 動詞の接続(アット)	接続詞 接続詞 接続詞	4.1 4.4 4.1
& アンド	u & v u &. v u &: v	Atop Under(Dual) Appose	動詞・名詞の接続(アトップ) 動詞の接続(アンダー) 動詞・名詞の接続(アトップ)	接続詞 接続詞 接続詞	4.1 4.4 4.1
? 疑問府	? Y X ? Y ?. Y X ?. Y	Roll Deal Roll(fix seed) Deal(fix seed)	重複を許した乱数の発生 重複を許さぬ乱数の発生 重複を許した乱数(シード固定) 重複を許さぬ乱数(シード固定)	動詞 動詞 動詞 動詞	1.6 1.6 1.6 1.6
A	A. Y X A. Y a. Y a: ad ar	Anagram Index Anagram Alphabet Ace ad ar	組合せ 辞書式順序 ｱﾙﾌｧﾍﾞｯﾄ等のｷｬﾗｸﾀｰ(256)の生成 「<''(Boxed Empty)」と同じ 度数での複素数表示 ラジアンでの複素数表示	副詞 動詞 名詞 名詞 名詞 名詞	1.8 1.8 5.5 5.5 5.5 5.5

B	b. Y	Boolean	ブール数へ変換	副詞	2.2
C	u b. n	Basic	ランク、逆関数、原始定義	副詞	5.6
	nbm	Character	mのｎ進数の１０進数の値	名詞	5.5
	C. Y	Cycle	巡回置換	動詞	1.8
	X C. Y	Permute	置換	動詞	1.8
D	u D. n	Derivative	多項式の微分係数を与える	接続詞	1.7
E	X D: Y	Secant Slope	平均変化率を与える	接続詞	1.7
	u d. n	Derivative	ランク０に固定「uD.n」と同じ	接続詞	1.7
	X E. Y	Member	ＹのＸを含む位置に１を与える	動詞	2.1
	e. Y	Raze in	アトムの含合積「-/～」と同じ	動詞	2.1
	X e. Y	Member of Int.	ＹがＸを含めば１(論理演算)	動詞	2.1
	x e n	Exponent	数値の指数表示	名詞	5.5
F	X f. Y	Fix	定義関数(代動詞)の固定	副詞	5.6
H	H. Y	Hypergeometric	超幾何級数	接続詞	
I	i. Y	Integer	負でない整数列(アレイ)の生成	動詞	3.1
	X i. Y	Index of	Ｙの要素のＸの位置インデクス	動詞	3.1
	i: Y				
J	j. Y	Imaginary	９０度回転した複素数を与える	動詞	1.4
L	X j. Y	Complex	複素数生成	動詞	1.4
I	m j n	Complex number	複素数	名詞	5.5
	L. Y	Level	ボックスのレベルを表示する	動詞	3.7
	uL:n Y	Level	ボックスのレベルでの演算	接続詞	3.7
N	m.	Left Noun	名詞の左引数	代用値	5.3
N	n.	Right Noun	名詞の右引数	代用値	5.3
O	NB.	Comment	注釈		5.5
	o. Y	Pi Times/	円周率(π)の倍率	動詞	1.3
	n o. Y	Circle Func.	円関数	動詞	1.3
P	p. Y	Polynomial	多項式	動詞	1.7
Q	p: Y	Prime	(Ｙ＋１)番目の素数を表示	動詞	1.6
R	mpn	Pi-times	円周率(パイ)「(mπ)n」	名詞	5.5
	q: Y	Prime Factors	素因数分解	動詞	1.6

	r. Y	Angle	Ｙを偏角にもつ単位複素数	動詞	1.4
	X r. Y	Polar	「r. Y」のＸ倍	動詞	1.4
	r	Rational	分数表示	名詞	5.5
T	T. Y	Taylor Approx.	テイラー展開による近似式	接続詞	1.7
	u t. Y	Taylor Coeff.	テイラー展開の係数	副詞	1.7
	u t: Y	Weighted T.C.	「u t: Y」＝「(!Y)*u t. Y」	副詞	1.7
U	u.	Left Verb	動詞の左引数	代用値	5.5
V	v.	Right Verb	動詞の右引数	代用値	5.5
X	mxn	Exponential	オイラーの定数「(ｍｅ)n」	名詞	5.5
Y	x:	Extended Precision	拡張精度での表示	名詞	5.5
0 〜 9	x.	Left Argument	左引数(一般用)	代用値	5.5
	y.	Right Argument	右引数(一般用)	代用値	5.5
	0:〜9: 〜_9:	Constant		動詞	5.5

【Ｊ言語　川柳・都々逸　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」
・「接続詞」　右にこだわる　接着剤
＊形無き　たった１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　リストが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　「セル」という
◇１つだけ　ランクの低い　セルだけ特に　「アイテム」という
◇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
◇ランクに等しい　セル指定すりゃ　つけずもがなの　副詞なり
☆左右が先で　中の動詞が　後で働く　これ「フォーク」（３連動詞）
☆二項動詞　片側動詞と　連なれば　引数取り込む　「フック」なり（２連動詞）
☆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　残った動詞と　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(＝：)で　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
・小にチョン(＜．)　切捨てご免で　整数値　大にチョン(＞．)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(＋：)は　所得倍増
・複素数　実部と虚部は　プラスピリ(＋．)　両側形なら　最大公約数
・大きさと　偏角求める　スターピリ(＊．)　両側形なら　最小公倍数
・スターコロン(＊：)は　平方値　パーセントコロン(％：)で　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　アイテム数は　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(｜．)　コロンつけたら　転置行列
・ボックスで　囲み連結　セミコロン(；)　片側形なら　ボックスはずし
・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　瞬時に計算　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　使えばいい

Ｊ言語のためのｸｲｯｸ･ﾚﾌｧﾚﾝｽ
《QUICK REFERENCE FOR J-LANGUAGE》

1998 12 12　第１版
著者　日本ＡＰＬ協会・Ｊ言語研究会
日本ＡＰＬ協会事務局　１４４－００２２品川区五反田１－８－１３
増島ビル　（株）プライド内
ＴＥＬ　０３－３２８０－０４１１
ＦＡＸ　０３－３２８０－０４１８

Ｊ言語の入手先：http://www.jsoftware.com　(Toront,Canada)
Ｊ言語のホームページ：http://www.soc.ae.keio.ac.jp/takeuchi/JAPLA
Ｊ言語研究会のメーリングリストへ登録：mitamura@ie.musashi-tech.ac.jp
J Quick Reference(PDF版）への意見：JCD02773@nifty.ne.jp
