Adám Brudzewsky

(material by Jay Foad)

大道至簡

Fun with Total Array Ordering (TAO)

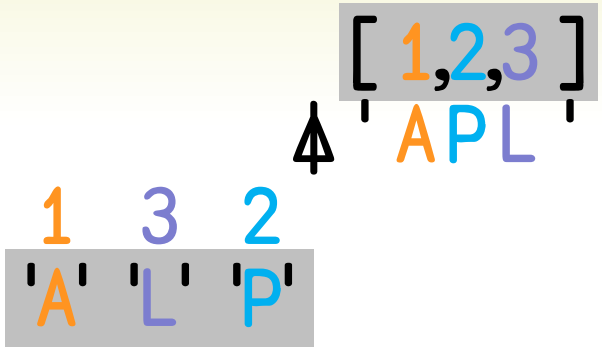
Grading

```
      ⍋'APL'
1 3 2
```

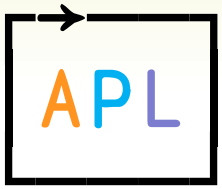
Grading

```
      ⍋'APL'
1 3 2
```

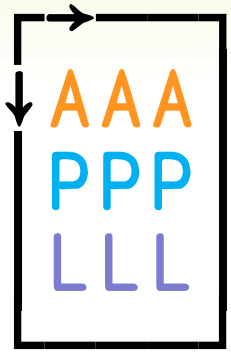
Grading



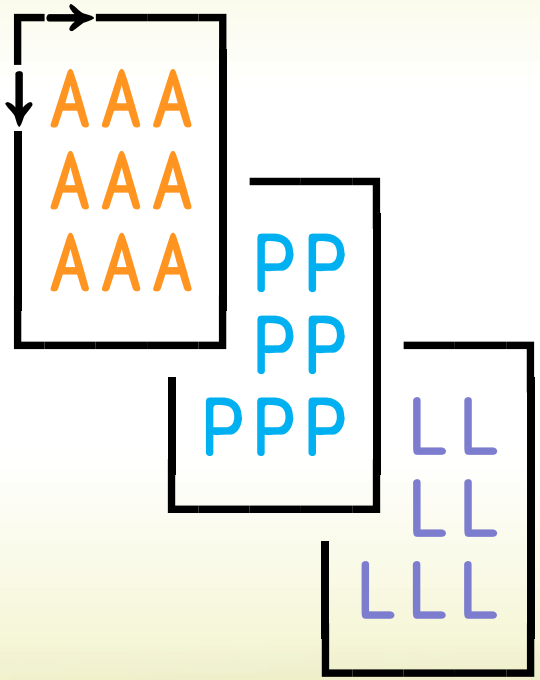
Major Cells: Vector



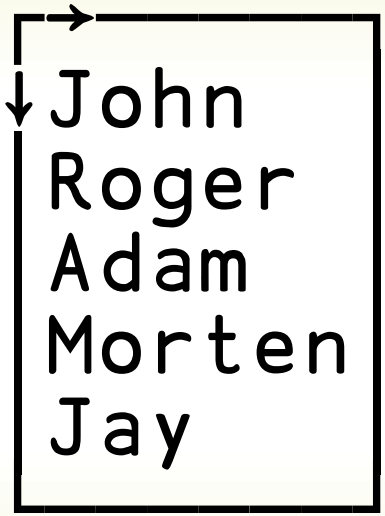
Major Cells: Matrix



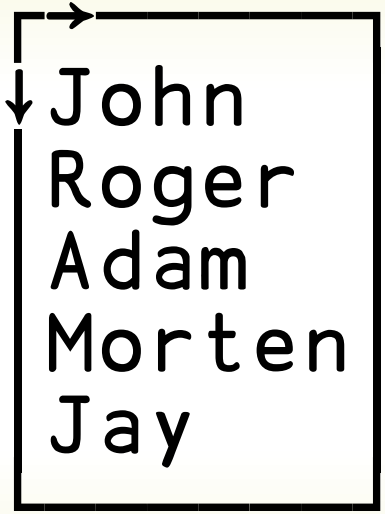
Major Cells: Cube



Major Cells: `mat`



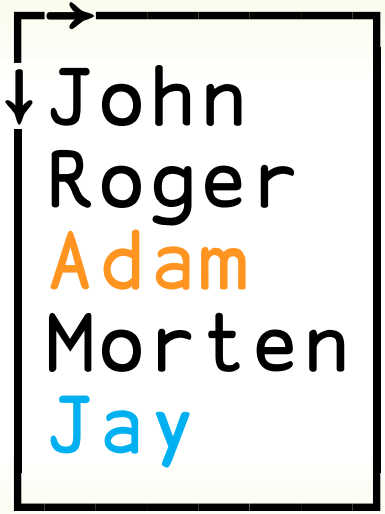
Major Cells: mat



⍋mat

3 5 1 4 2

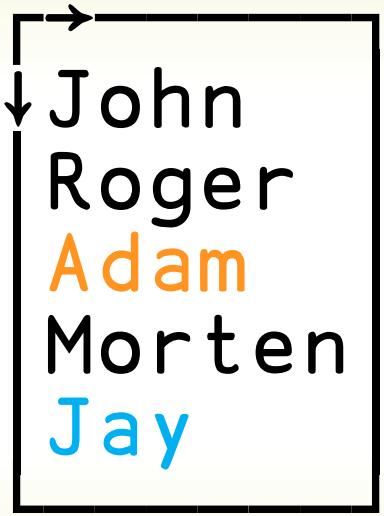
Major Cells: mat

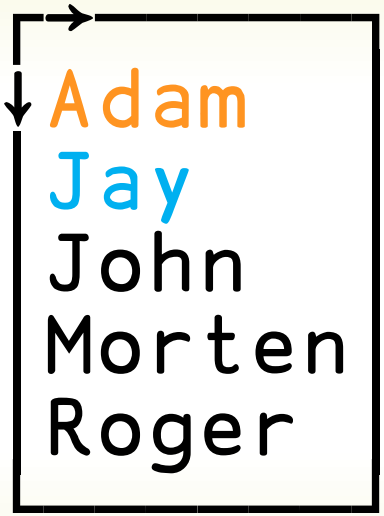


⍋mat

3 5 1 4 2

Major Cells: `mat` `mat[⍋mat;]`





Sorting

```
mat[⍋mat;]
```

```
{(⊂⍋⍵)⌷⍵} mat
```

Sorting

```
mat[⍋mat;]
```

```
{(⊂⍋⍵)⌷⍵} mat
```

Sorting

```
mat[⍋mat;]
```

```
{(⊂⍋⍵)⌷⍵} mat
```

Sorting

```
mat[⍋mat;]
```

```
Sort ← {(⊂⍋⍵)⌷⍵}
```

Sorting

```
      Sort ← {(⊂⍋⍵)⌷⍵}
      Sort mat
```

```
mat[⍋mat;]
```

Sorting

```
      mat[⍋mat;]
      Sort ← {(⊂⍋⍵)⌷⍵}
      Sort mat
Adam
Jay
John
Morten
Roger
```

Sorting

```
      n ← 'Roger' 'Adam' 'Jay'
```

Sorting

```
      n ← 'Roger' 'Adam' 'Jay'

      Sort n    ⍝ 16.0
```

Sorting

```
      n ← 'Roger' 'Adam' 'Jay'

      Sort n    ⍝ 16.0
DOMAIN ERROR
```

Sorting

```
      n ← 'Roger' 'Adam' 'Jay'

      Sort n    ⍝ 16.0
DOMAIN ERROR

      Sort n    ⍝ 17.0
```

Sorting

```
      n ← 'Roger' 'Adam' 'Jay'

      Sort n    ⍝ 16.0
DOMAIN ERROR

      Sort n    ⍝ 17.0
```

Adam	Jay	Roger

A bit of history

```
'Morten'
'Roger '
```

A bit of history

```
'Morten'
'Roger '
```

A bit of history

```
A ← 'Morten'
B ← 'Roger '
```

A bit of history

```
      A ← 'Morten'
      B ← 'Roger '

      ⍋ A ,[0.5] B
1 2
```

A bit of history

```
      A ← 'Morten'
      B ← 'Roger '

      ⍋ A ,[0.5] B
1 2

  'Morten' ≺ 'Roger '
```

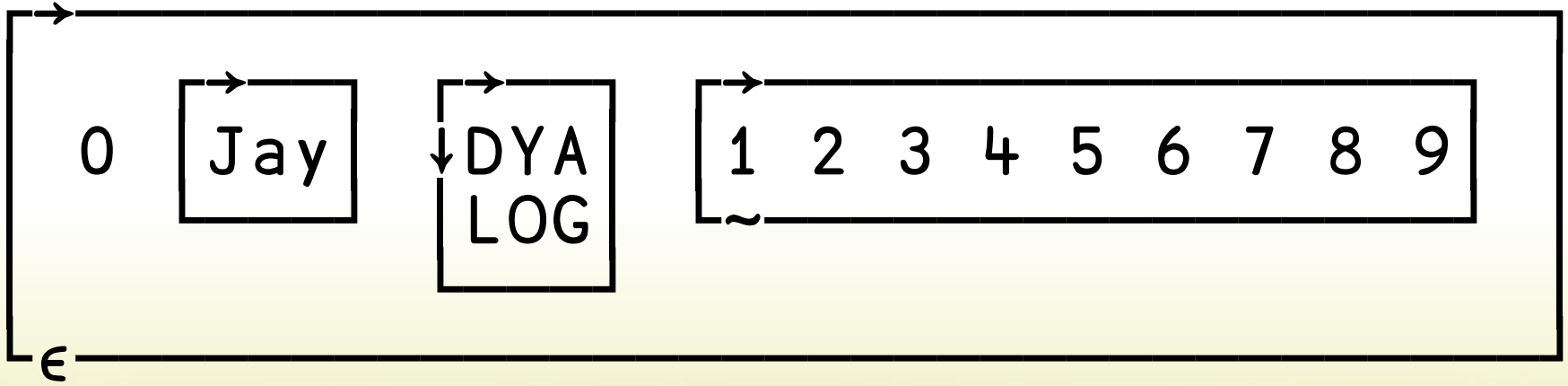
Total Array Ordering

some array ≺ other array ?

Total Array Ordering

How would you order:

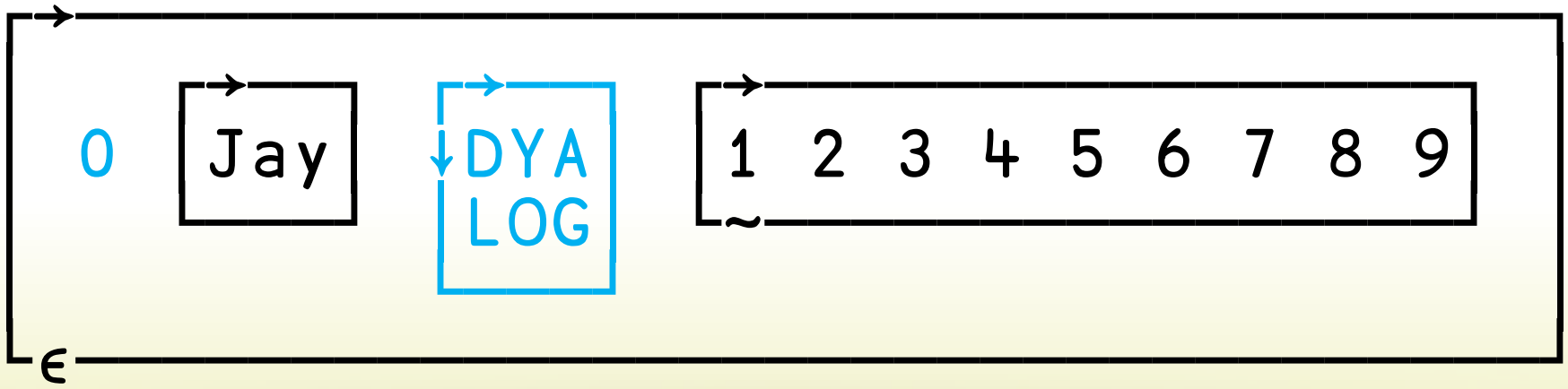
```
      ⍞ ← 0 'Jay' (2 3⍴'DYALOG') (⍳9)
```



Total Array Ordering

How would you order:

```
      ⍋ ← 0 'Jay' (2 3⍴'DYALOG') (⍳9)
```



Total Array Ordering

TAO

TAO

Total order

Page issues

In mathematics, a **linear order**, **total order**, **simple order**, or **(non-strict) ordering** is a binary relation on some set X, which is antisymmetric, transitive, and total (this relation is denoted here by infix $\leq$). A set paired with a total order is called a **totally ordered set**, a **linearly ordered set**, a **simply ordered set**, or a **chain**.

More symbolically, a set X is totally ordered under $\leq$ if the following statements hold for all a, b and c in X:

If $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetry);
If $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity);
$a \leq b$ or $b \leq a$ (totality).

TAO

some array ≺ other array ?

TAO

some array ≺ other array ?

TAO

some array
no refs

other array
no refs

?

TAO Rules

TAO Rules

Compatibility: simple arrays

TAO Rules

Compatibility: simple arrays

Consistency: $(\alpha \prec \omega) \wedge (\omega \prec \alpha) \Leftrightarrow \alpha = \omega$

TAO Rules

Compatibility: simple arrays

Consistency: $(\alpha \prec \omega) \wedge (\omega \prec \alpha) \Leftrightarrow \alpha = \omega$

$\alpha \equiv \omega$

TAO Rules

Compatibility: simple arrays

Consistency: $(\alpha \prec \omega) \wedge (\omega \prec \alpha) \Leftrightarrow \alpha = \omega$

$\alpha \equiv \omega$

$\theta \neq ' '$

TAO Rules: simple scalars

TAO Rules: simple scalars

character vs character: ✓

TAO Rules: simple scalars

character vs character: ✓

real number vs real number: ✓

TAO Rules: simple scalars

character vs character: ✓

real number vs real number: ✓

complex: `1 ≺ 1J1 ≺ 2J¯1 ≺ 2`

TAO Rules: simple scalars

character vs character: ✓

real number vs real number: ✓

complex: `1 ≺ 1J1 ≺ 2J¯1 ≺ 2`

number vs character: `100 ≺ 'A'`

TAO Rules: simple scalars

character vs character: ✓

real number vs real number: ✓

complex: `1 ≺ 1J1 ≺ 2J¯1 ≺ 2`

number vs character: `100 ≺ 'A'`

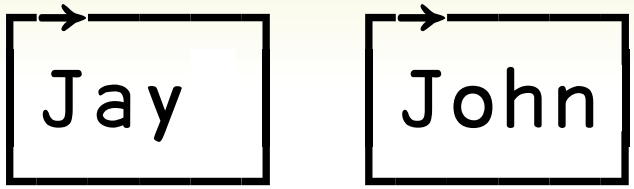
null: `⍬NULL ≺ ¯100`

TAO Rules: simple scalars

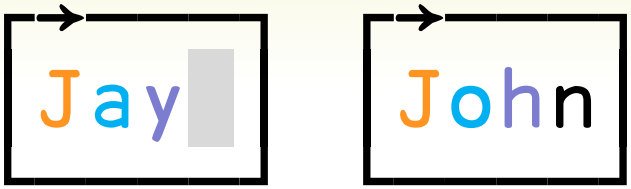
```
⍬NULL ≺ ¯1 ≺ 0J¯2 ≺ 0 ≺ 0J2 ≺ 1 ≺ 'A'
```

TAO Rules: same shape

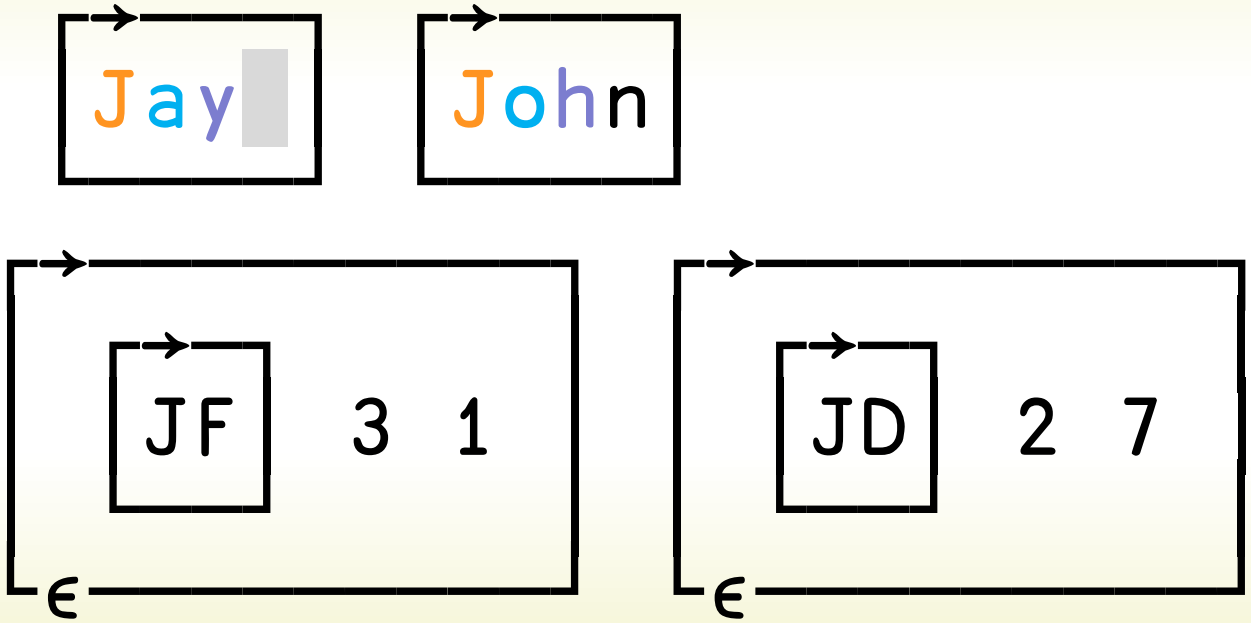
TAO Rules: same shape



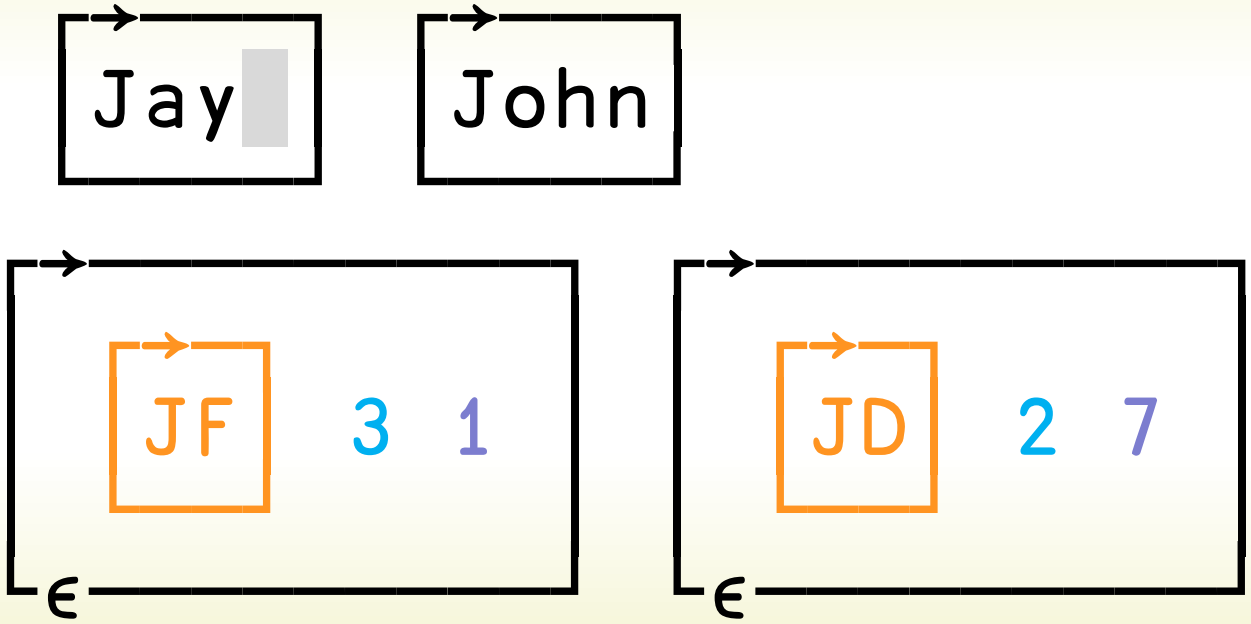
TAO Rules: same shape



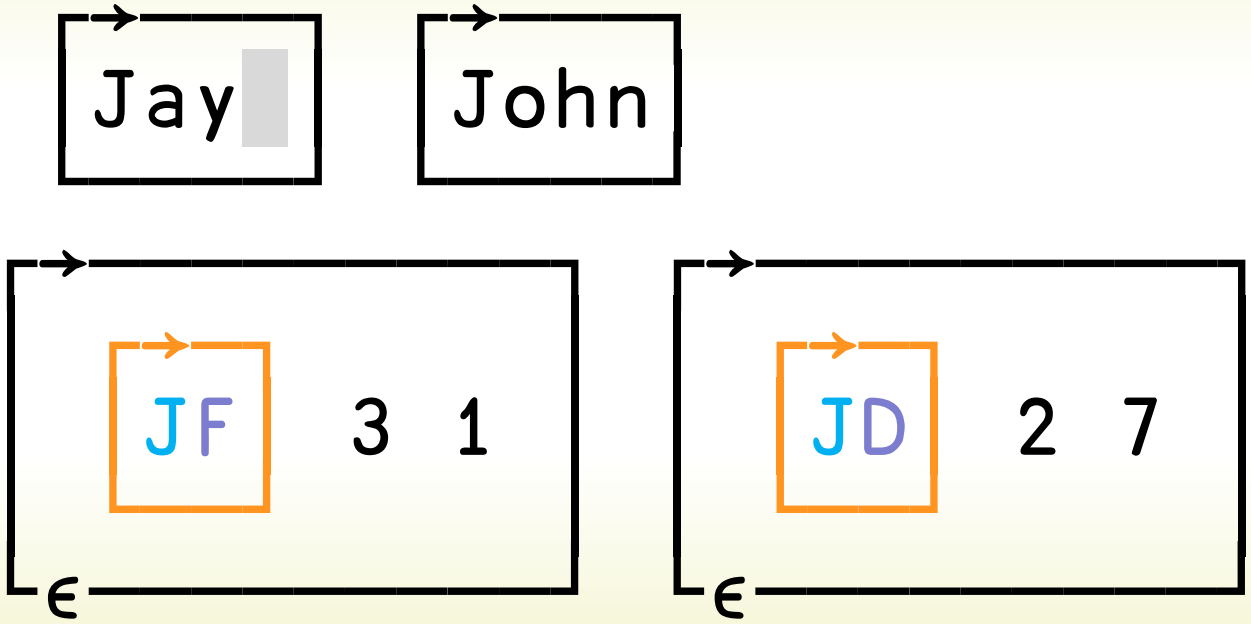
TAO Rules: same shape



TAO Rules: same shape



TAO Rules: same shape



TAO Rules: different shape

```
CAN

CAR

CARPENTER

CARPET

CAT
```

TAO

Lexicographical order

For similarly named ordering systems outside mathematics, see alphabetical order, natural sort order, lexicographic preferences, and collation.

In mathematics, the **lexicographic** or **lexicographical order** (also known as **lexical order**, **dictionary order**, **alphabetical order** or **lexicographic(al) product**) is a generalization of the way words are alphabetically ordered based on the alphabetical order of their component letters. This generalization consists primarily in defining a total order over the sequences (often called strings in computer science) of elements of a finite totally ordered set, often called alphabet.

TAO Rules: different shape

```
CAR

CARPENTER
```

TAO Rules: different shape

CAR

CARPENTER

TAO Rules: different shape

CAR 

CARPENTER

TAO Rules: different shape

```
CAN

CAR

CARPENTER

CARPET

CAT
```

TAO Rules: different shape

```
CAN

CAR

CARPENTER

CARPET

CAT
```

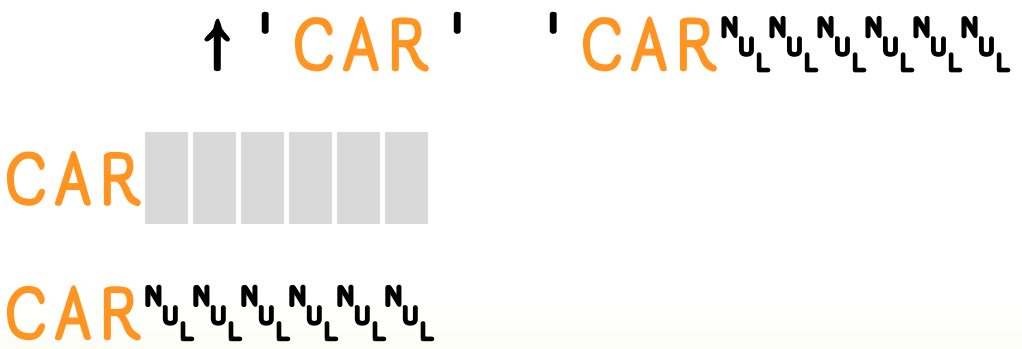
TAO Rules: different shape

```
↑'CAR' 'CARPENTER'
```

TAO Rules: different shape

```
      ↑'CAR' 'CARPENTER'
CAR
CARPENTER
```

TAO Rules: different shape



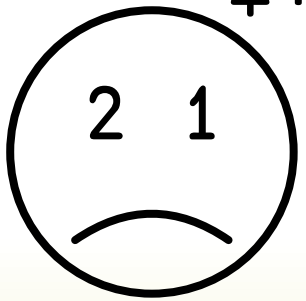
TAO Rules: different shape

```
      ⍋↑'CAR'  'CAR⍬⍬⍬⍬⍬⍬'

2 1
```

TAO Rules: different shape

```
      ⍋⍋ 'CAR'   'CAR⍬⍬⍬⍬⍬⍬'
```



TAO Rules: different shape

1 2 3

1 2 3 4

TAO Rules: different shape

```
¯1 ¯2 ¯3

¯1 ¯2 ¯3 ¯4
```

TAO Rules: different shape

```
¯1 ¯2 ¯3
¯1 ¯2 ¯3 ¯4
```

TAO Rules: different shape

```
↑(¯1 ¯2 ¯3) (¯1 ¯2 ¯3 ¯4)
```

TAO Rules: different shape

```
      ↑(¯1 ¯2 ¯3) (¯1 ¯2 ¯3 ¯4)
¯1 ¯2 ¯3  0
¯1 ¯2 ¯3 ¯4
```

TAO Rules: different shape

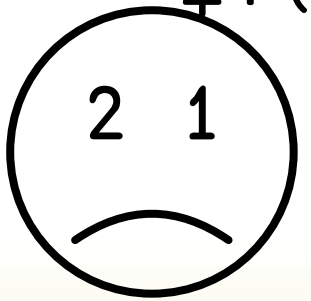
```
⍋↑(¯1 ¯2 ¯3) (¯1 ¯2 ¯3 ¯4)
```

TAO Rules: different shape

```
      ⍋↑(¯1 ¯2 ¯3) (¯1 ¯2 ¯3 ¯4)
2 1
```

TAO Rules: different shape

```
      ⍋(¯1 ¯2 ¯3) (¯1 ¯2 ¯3 ¯4)
```



TAO Rules: different shape

TAO Rules: different shape

```
      ¯∞ ≺ ⍬ ≺ NULL ≺ ¯1 ≺ 0J¯2 ≺ 0 ≺ 0J2 ≺ 1 ≺ 'A'
```

TAO Rules: different shape

CAR

CARPENTER

TAO Rules: different shape

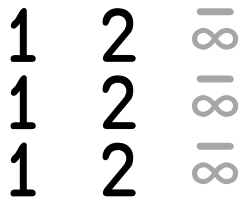
CARPENTER

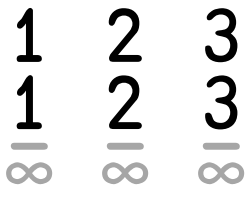
TAO Rules: different shape

```
1 2
1 2
1 2
```

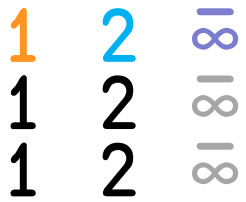
```
1 2 3
1 2 3
```

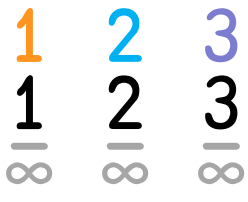
TAO Rules: different shape





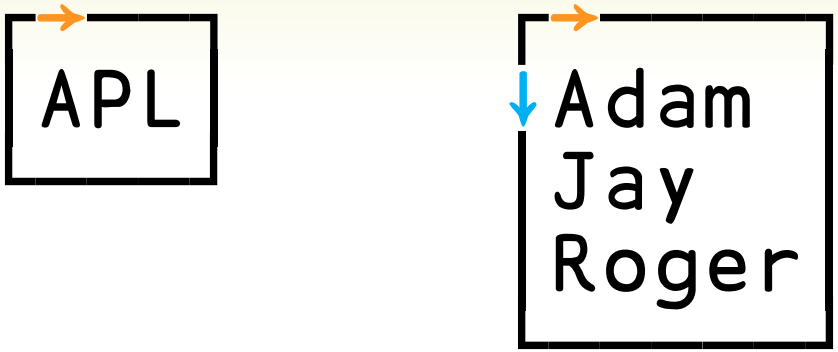
TAO Rules: different shape



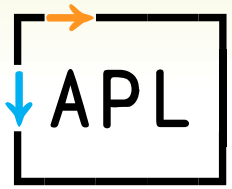


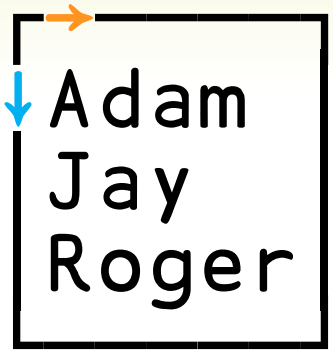
TAO Rules: different shape

TAO Rules: different rank

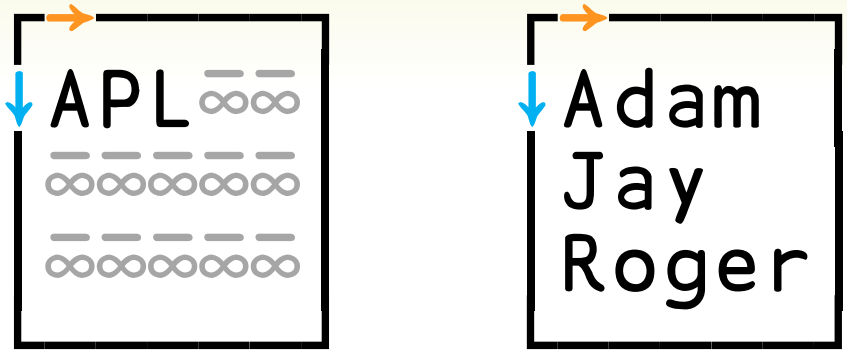


TAO Rules: different rank

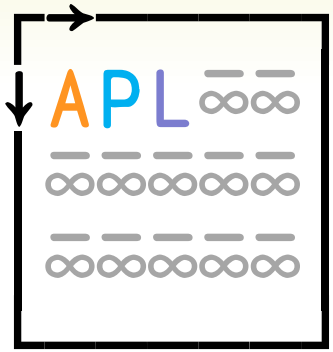


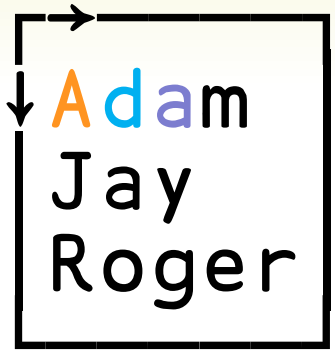


TAO Rules: different rank

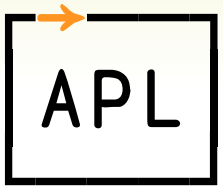


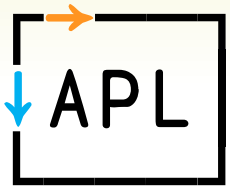
TAO Rules: different rank



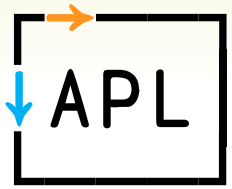


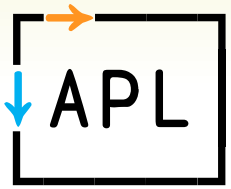
TAO Rules: different rank



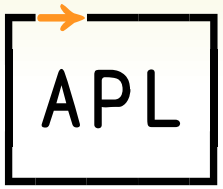


TAO Rules: different rank

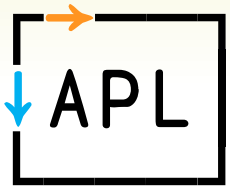




TAO Rules: different rank



1



2

TAO Rules: different rank

scalar

```
   'A'
A
```

vector

```
   ,'A'
A
```

vector

```
   3⍴'APL'
APL
```

1-row matrix

```
   1 3⍴'APL'
APL
```

TAO Rules: recap

simple scalars	⍬ NULL ≺ nums ≺ chars
same shape	item-by-item in ravel order
unequal shape	pad with ¯∞
unequal rank	prefix shape with 1s

TAO Rules: empty arrays

```
      2 0 0⍴⍬                0 0 3⍴''
```

TAO Rules: empty arrays

```
2 0 0⍴⍬          0 0 3⍴''
```

TAO Rules: empty arrays

```
2 0 0⍴⍬        0 0 3⍴''
2 0 3⍴⍬        2 0 3⍴''
```

TAO Rules: empty arrays

```
      2 0 0⍴⍬                    0 0 3⍴''
0 0 3

⍬ prototypical items ''
or
2 0 0 original shapes 0 0 3
?
```

TAO Rules: empty arrays

```
2 0 0⍴⍬          0 0 3⍴''
```

⍬ prototypical items ''

2 0 0 original shapes 0 0 3

TAO Rules: empty arrays

2 0 0⍴⍬ 0 0 3⍴⍬

⍬ prototypical items ⍬

2 0 0 original shapes 0 0 3

TAO Rules

simple scalars	⍬ NULL ≺ nums ≺ chars
same shape	item-by-item in ravel order
unequal shape	pad with ¯∞
unequal rank	prefix shape with 1s
both empty	type then shape

TAO Scope

⍋⍵

⍒⍵

⍺⍳⍵

Demo

Features coming in version 17.0...

Demo

Turing Awards

```
]load /tao/turing
Sort ← {(⊂⍋⍵)⌷⍵}
Sort turing
vol ← ⊣/5 5⍴⎕A
Cat ← {vol⍪⍵}
Chop ← {(Cat ⍵){⊂⍵}⌸ ⍵}
Chop turing
Sort Chop turing
```

ISO 8601

yyyy-mm-ddThh:mm:ss.mmm

Stock Tickers

```
]load /tao/stocks
Sort stocks
Under←{α←⊢ ⋄ (⍵⍵⍣¯1) α ⍺⍺ ⍵⍵ ⍵}
Sort Under(1⌽⊢) stocks
2⌷Under⍉ stocks
⍋ 2⌷Under⍉ stocks
Puncts ← {~⍵∊⎕D}
Nums ← ⍕¨ '@Puncts
Nums¨ 2⌷Under⍉ stocks
⍋ Nums¨ 2⌷Under⍉ stocks
Nums¨@2 Under⍉ stocks
⍋ Nums¨@2 Under⍉ stocks
⍋ stocks
```

Demo

Phonebook Record Insertion

```
]load /tao/pb
Sort ← {(⊂⍋⍵)⌷⍵}
Sort pb
pb ← Sort pb
new ← 'Kromberg' 'Morten' 584
pb ⍳ new
(2↑pb),⊂new,(2↓pb)
2(↑,⊂new,↓)pb
Into ← {(⍵⍳⍺)(↑,⊂⍺,↓)⍵}
new Into pb
new2←'Becker' 'Brian' 563
⊃Into/new new2 pb
```

Spreadsheet Data Types

```
ss ← (⎕NEW⊂'Clipboard').Array
Sort ss
ss[⍋ss[;1 3];]
```

Natural Sorting

```
)clear
]load /tao/names2
num←'((¯?\d+\.?\d*|¯?\d*\.\d+)(E¯?\d+)?)(J(?1))?'
ExecNums ← {⍵.PatternNum:⍵.Match ⋄ ⍎⍵.Match}
Order ← {⍋(∈num'.'⎕S ExecNums⍠1)¨⍵}
Order names
SuperNatSort ← {(⊂Order ⍵)⌷⍵}
SuperNatSort names
]defs -topdown
```

TAO: a true team effort

Arthur Whitney	initial suggestion
Roger Hui	TAO of J
John Scholes	model: dfns.dyalog.com/n_le.htm
Roger Hui	Dyalog implementation

Adám Brudzewsky, Roger Hui, Jay Foad,
John Scholes, Morten Kromberg

TAO: a true team effort

Essays/The TAO of J

< Essays

The TAO (total array ordering) of J is, from the /: page of the dictionary:

The types: numeric or empty, symbol, literal (1 byte or 2 byte characters), and boxed, are so ordered; within them, a lower rank precedes a higher, and arrays to be compared are padded with fills if necessary to have the same shape. Complex arguments are ordered by real part, then by imaginary. Boxed arrays are ordered according to the opened elements.

A set of relations that are consistent with the comparison used in the standard sort can be defined simply:

```
ge=: 0 1 -: \:@,&<
gt=: 1 0 -: /:@,&<
le=: 0 1 -: /:@,&<
lt=: 1 0 -: \:@,&<
eq=: -:!.0
```

TAO: a true team effort

Arthur Whitney	initial suggestion
Roger Hui	TAO of J
John Scholes	model: dfns.dyalog.com/n_le.htm
Roger Hui	Dyalog implementation

Adám Brudzewsky, Roger Hui, Jay Foad,
John Scholes, Morten Kromberg

TAO: a true team effort

DYALOG

```
precedes ← this ##.le that          ⍝ Total array ordering (TAO) comparison.

Inspired by Roger Hui,  [le]  "less than or equal"  defines a Total Order on APL
arrays. For any arrays A, B and C:

    (A le B) ∧ (B le A) : A eq B    ⍝ antisymmetry
    (A le B) ∧ (B le C) : A le C    ⍝ transitivity
    (A le B) ∨ (B le A)             ⍝ totality.

See: http://en.wikipedia.org/wiki/Total_order
See: http://www.jsoftware.com/jwiki/Essays/The_TAO_of_J

Given any two arrays, [le]  tells  us which "precedes or is equal to" the other.
In particular it may be used to compare arrays of:

    differing type:     '6' le 6
    differing rank:     3 le ,3
    differing shape:    5 4 le 5 4 3
    differing depth:    (⊂⍳2)le ⊂⊂⍳2
    complex numbers:    7j8 le 8j7
```

TAO: a true team effort

Arthur Whitney	initial suggestion
Roger Hui	TAO of J
John Scholes	model: dfns.dyalog.com/n_le.htm
Roger Hui	Dyalog implementation

Adám Brudzewsky, Roger Hui, Jay Foad,
John Scholes, Morten Kromberg

Features coming in version 17.0

Grade and Sort any[no refs] Array

`{(⊂⍋⍵)⌷⍵}` for all ranks – *fast!*

`⍋⍵` `⍒⍵` `⍺⍸⍵`