DYALOG

Belfast 2018

```
(Title:['Array     '
        'Notation'
        'Mk III   ']

 Presenter:('Adám'
            'Brudzewsky'))
```

adam@dyalog.com

DYALOG

Belfast 2018

```
(Title:['Array    '
        'Notation'
        'Mk III   ']

 Presenter:('Adám'
            'Brudzewsky'))
```

adam@dyalog.com

DYALOG

Belfast 2018

```
(Title:['Array    '
        'Notation'
        'Mk III   ']

Presenter:('Adám'
           'Brudzewsky'))
```

adam@dyalog.com

DYALOG

Belfast 2018

```
(Title:['Array'
        'Notation'
        'Mk III'  ]

Presenter:('Adám'
           'Brudzewsky'))
```

adam@dyalog.com

DYALOG

Belfast 2018

```
(Title:['Array'
        'Notation'
        'Mk III'  ]

 Presenter:('Adám'
            'Brudzewsky'))
```

adam@dyalog.com

DYALOG

Belfast 2018

```
(Title:['Array'
        'Notation'
        'Mk III'   ]
```

Presenter:('Adám' ◇ 'Brudzewsky'))

adam@dyalog.com

DYALOG

Belfast 2018

```
(Title:['Array'
        'Notation'
        'Mk III'  ]

Presenter:('Adám' ◇ 'Brudzewsky'))
```

adam@dyalog.com

DYALOG

Belfast 2018

```
(Title:['Array'
        'Notation'
        'Mk III']
 Presenter:('Adám' ◇ 'Brudzewsky'))
```

adam@dyalog.com

DYALOG

Belfast 2018

```
(Title:[
          'Array'
          'Notation'
          'Mk III'
        ]
 Presenter:('Adám' ◇ 'Brudzewsky'))
```

adam@dyalog.com

DYALOG

Belfast 2018

```
(Title:[
          'Array'
          'Notation'
          'Mk III'
        ]
 Presenter:('Adám' ◇ 'Brudzewsky'))
```

adam@dyalog.com

DYALOG

Belfast 2018

```
(Title:[
          'Array'
          'Notation'
          'Mk III'
        ]
 Presenter:('Adám' ◇ 'Brudzewsky'))
```

adam@dyalog.com

Mk III?


2015
2016
2017
2018
2019
Phil Last
Sicily
Dyalog Mk I
internal
Dyalog Mk II
Elsinore
Dyalog Mk III
Belfast
Acre-3 Mk I
Acre-4 Mk I
Acre-4 Mk II
?

Mk III?


2015
2016
2017
2018
2019
Phil Last
Sicily
Dyalog Mk I
internal
Dyalog Mk II
Elsinore
Dyalog Mk III
Belfast
Acre-3 Mk I
Acre-4 Mk I
Acre-4 Mk II
?

Mk III?


2015
2016
2017
2018
2019
Phil Last
Sicily
Dyalog Mk I
internal
Dyalog Mk II
Elsinore
Dyalog Mk III
Belfast
Acre-3 Mk I
Acre-4 Mk I
Acre-4 Mk II
?

Mk III?


2015
2016
2017
2018
2019
Phil Last
Sicily
Dyalog Mk I
internal
Dyalog Mk II
Elsinore
Dyalog Mk III
Belfast
Acre-3 Mk I
Acre-4 Mk I
Acre-4 Mk II
?

Mk III?


2015
2016
2017
2018
2019
Phil Last
Sicily
Dyalog Mk I
internal
Dyalog Mk II
Elsinore
Dyalog Mk III
Belfast
Acre-3 Mk I
Acre-4 Mk I
Acre-4 Mk II
?

Mk III?


2015
2016
2017
2018
2019
Phil Last
Sicily
Dyalog Mk I
internal
Dyalog Mk II
Elsinore
Dyalog Mk III
Belfast
Acre-3 Mk I
Acre-4 Mk I
Acre-4 Mk II
?

Mk III?


2015
2016
2017
2018
2019
Phil Last
Sicily
Dyalog Mk I
internal
Dyalog Mk II
Elsinore
Dyalog Mk III
Belfast
Acre-3 Mk I
Acre-4 Mk I
Acre-4 Mk II
?

Mk III?


2015
2016
2017
2018
2019
Phil Last
Sicily
Dyalog Mk I
internal
Dyalog Mk II
Elsinore
Dyalog Mk III
Belfast
Acre-3 Mk I
Acre-4 Mk I
Acre-4 Mk II
?

Mk III?


2015
2016
2017
2018
2019
Phil Last
Sicily
Dyalog Mk I
internal
Dyalog Mk II
Elsinore
Dyalog Mk III
Belfast
Acre-3 Mk I
Acre-4 Mk I
Acre-4 Mk II
?

Array Notation

We have good notations for

- simple scalars and vectors
- small, depth-2 nested arrays

We need notations for

- higher rank arrays
- more complex nested arrays

Why?

Array "Notation"

Array "Notation"

```
poss←1 2⍴'fns'((0 1)(0.7 0)(0.7 0)×size)
poss,←'fnd'((0 1)(0 0)(0 0)×size)
poss,←'lines'((0 0)(0.7 0)(0.7 0)×size)
poss,←'lnd'((0 0)(0 0)(0 0)×size)
```

Array "Notation"

```
Q1←'January' 'February' 'March    '~¨' '    ⍝ 1st quarter month names.
Q2←'April  ' 'May      ' 'June     '~¨' '    ⍝ 2nd   ..       ..     ..
Q3←'July   ' 'August   ' 'September'~¨' '    ⍝ 3rd   ..       ..     ..
Q4←'October' 'November' 'December '~¨' '    ⍝ 4th   ..       ..     ..
months←Q1,Q2,Q3,Q4                          ⍝ month names for year.
```

Array "Notation"

```
args←'zheev_'assoc↓⍉↑⍬{                ⍝ associate external fn.
      ⍵,⊂' <C1    ' 'V'}{               ⍝ JOBZ
      ⍵,⊂' <C1    ' 'L'}{               ⍝ UPLO
      ⍵,⊂' <I4    'n}{                  ⍝ N
      ⍵,⊂' =F8[]  '(⍴⍉mat)}{            ⍝ A
      ⍵,⊂' <I4    'n}{                  ⍝ LDA
      ⍵,⊂' >F8[]  'n}{                  ⍝ W
      ⍵,⊂' >F8[]  '(¯2+4×n)}{           ⍝ WORK
      ⍵,⊂' <I4    '(¯1+2×n)}{           ⍝ LWORK
      ⍵,⊂' >F8[]  '(¯2+3×n)}{           ⍝ RWORK
      ⍵,⊂' >I4    ' 0}⍬                 ⍝ INFO
```

Array "Notation"

```
morse←{                          ⍝ A Conversion to/from Morse code.
    P M←{⍵~¨' '}⍵⌿⍨0≠⍴¨⍵          ⍝ A plain-text and Morse codes.
        ('A' '.-'   )('B' '-...' )('C' '-.-.' )('D' '-..'  ),⍵}{
        ('E' '.'    )('F' '..-.' )('G' '--.'  )('H' '....' ),⍵}{
        ('I' '..'   )('J' '.---' )('K' '-.-'  )('L' '.-..' ),⍵}{
        ('M' '--'   )('N' '-.'   )('O' '---'  )('P' '.--.' ),⍵}{
        ('Q' '--.-' )('R' '.-.'  )('S' '...'  )('T' '-'    ),⍵}{
        ('U' '..-'  )('V' '...-' )('W' '.--'  )('X' '-..-' ),⍵}{
        ('Y' '-.--' )('Z' '--..' ),⍵}{

        ('0' '-----')('1' '.----')('2' '..---')('3' '...--'),⍵}{
        ('4' '....-')('5' '.....')('6' '-....')('7' '--...'),⍵}{
        ('8' '---..')('9' '----.'),⍵}{

        ('.' '.-.-.-')(',' '--..--')(':' '---...'),⍵}{
        ('?' '..--..')('¡' '.----.')('-' '-....-'),⍵}{
        ('/' '-..-.' )('(' '-.--.-')(')' '-.--.-'),⍵}{
        ('"' '.-..-.')('@' '.--.-.')('=' '-...-' ),⍵}{

        ⍵}⊂' ' ' / '             ⍝ A blank / inter-word separator.

    1=⌊≡,⍵:M[P⍳⍵∩P]              ⍝ A plain text to Morse.
    2=⌊≡,⍵:P[M⍳⍵∩M]              ⍝ A Morse to plain text.
}
```

Array Notation

```
]Boxing on -style=max
```

What we have

- Simple scalars

```
42
'a'
```

- Simple vectors

```
1 2 3
'Hello'
```

- Small vectors of vectors

```
(1 2 3) (4 5 6)
'Hello' 'World'
```

What we need

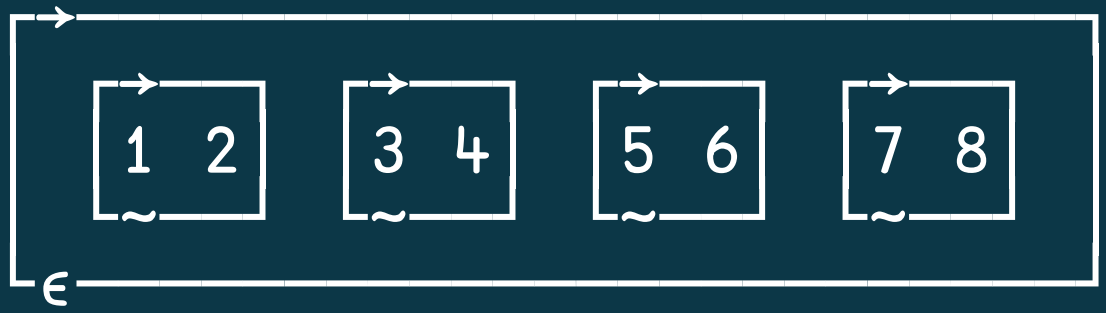
- More complex nested arrays

```
(1 2 3 'Hello'
 4 5 6 'World')
```

- Higher rank arrays

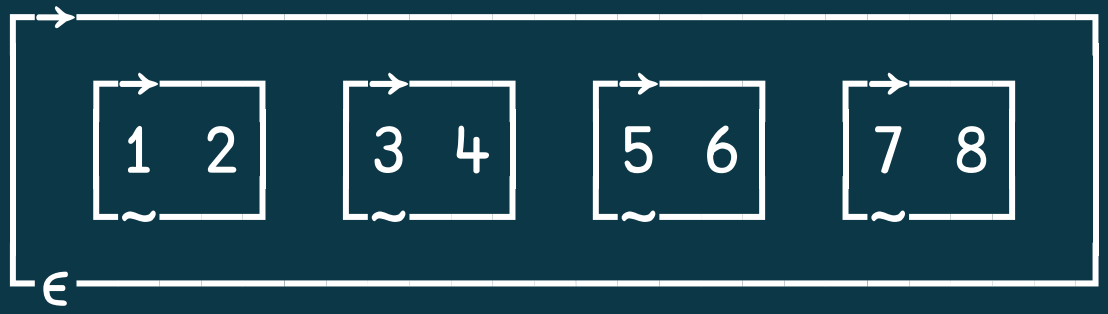
```
[1 2 3
 4 5 6]
```

Vector of Vectors




```
(1 2) (3 4) (5 6) (7 8)
```

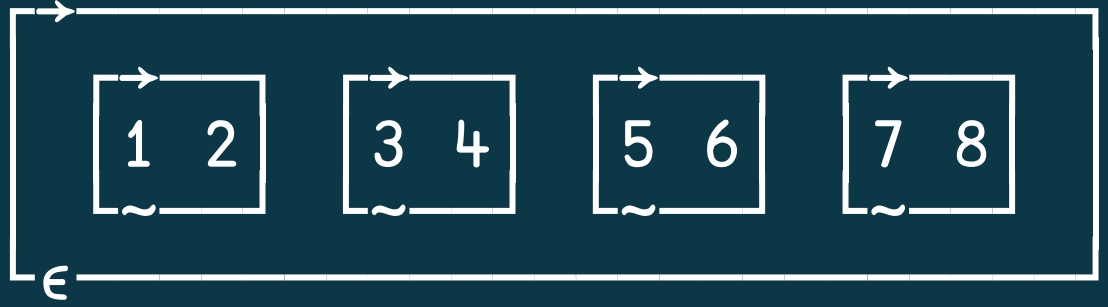
Vector of Vectors




```
(1 2
 3 4
 5 6
 7 8)
```

```
(1 2) (3 4) (5 6) (7 8)
```

Vector of Vectors

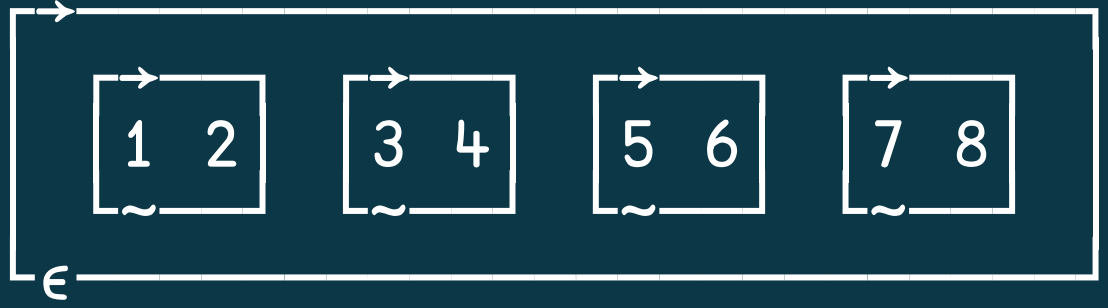



```
(1  2
 3  4
 5  6
 7  8)
```

```
(1  2) (3  4) (5  6) (7  8)
```

```
(1  2 ◇ 3  4 ◇ 5  6 ◇ 7  8)
```

Vector of Vectors



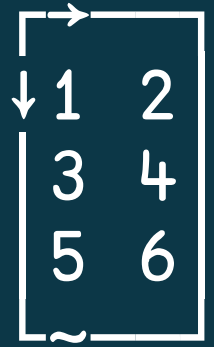

```
(1 2
 3 4
 5 6
 7 8)
```

```
(1 2) (3 4) (5 6) (7 8)
```

```
(1 2 ◇ 3 4 ◇ 5 6 ◇ 7 8)
```

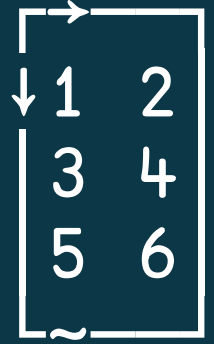
```
(1 2 ◇ 3 4
 5 6 ◇ 7 8)
```

Matrix



```
[1  2
 3  4
 5  6]
```

Matrix




```
[1  2
 3  4
 5  6]
```

```
[1  2 ◇ 3  4 ◇ 5  6]
```

Simple numeric matrix

Current

```
m←1 2⍴34 ¯12
m⍪←  43  ¯3
m⍪←   0   1.5
```

Proposed

```
m←[34 ¯12
   43  ¯3
    0   1.5]
```

```
┌→──────┐
↓34 ¯12 │
 43  ¯3 │
  0   1.5
└~──────┘
```

Simple character matrix

Current

```
r←1 5⍴'Three'
r⍪←   'Blind'
r⍪←   'Mice '
```

Proposed

```
r←['Three'
   'Blind'
   'Mice']
```

```
┌→┐
↓Three
 Blind
 Mice
└─┘
```

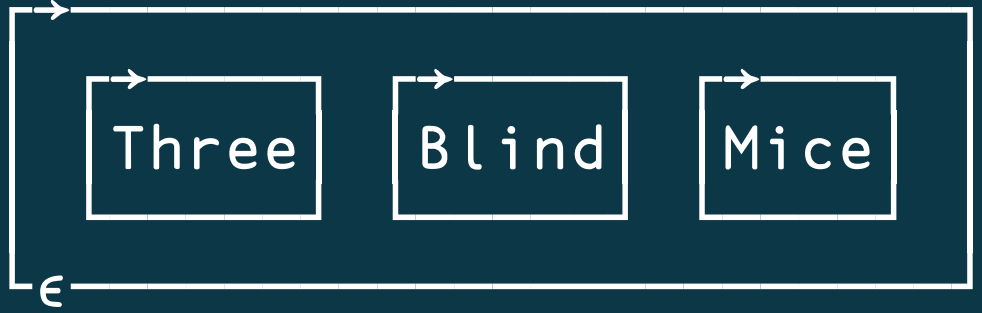
Vector of Text Vectors

Current

```
r← ⊂'Three'
r,←⊂'Blind'
r,←⊂'Mice'
```

Proposed

```
r←('Three'
   'Blind'
   'Mice')
```



Game of Life Pattern

Current

```
r←⊖⍪0 0 1 0 0
r⍪← 1 0 1 0 0
r⍪← 0 1 1 0 0
r⍪← 0 0 0 0 0
r⍪← 0 0 0 0 0
```

```
┌→──────────┐
↓0 0 1 0 0  │
 1 0 1 0 0  │
 0 1 1 0 0  │
 0 0 0 0 0  │
 0 0 0 0 0  │
└~──────────┘
```

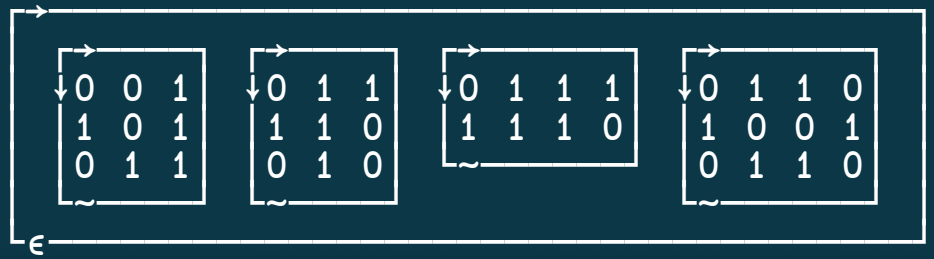
Proposed

```
r←[0 0 1 0 0
   1 0 1 0 0
   0 1 1 0 0
   0 0 0 0 0
   0 0 0 0 0]
```

Game of Life Patterns

Current

```
_←⊖⍤¯1⊢0 0 1
_⍪← 1 0 1
_⍪← 0 1 1
r←⊂_
_←⊖⍤¯1⊢0 1 1
_⍪← 1 1 0
_⍪← 0 1 0
r,←⊂_
_←⊖⍤¯1⊢0 1 1 1
_⍪← 1 1 1 0
r,←⊂_
_←⊖⍤¯1⊢0 1 1 0
_⍪← 1 0 0 1
_⍪← 0 1 1 0
r,←⊂_
⎕EX '_'
```



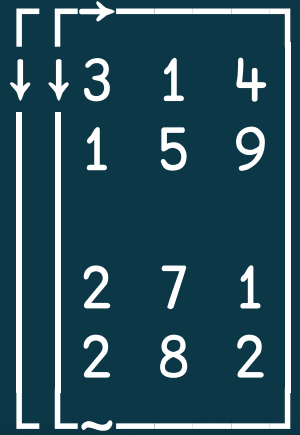
Proposed

```
r←([0 0 1
    1 0 1
    0 1 1]

   [0 1 1
    1 1 0
    0 1 0]

   [0 1 1 1
    1 1 1 0]

   [0 1 1 0
    1 0 0 1
    0 1 1 0])
```

Simple numeric 3D array

Current

```
d←1 2 3⍴3 1 4 1 5 9
d,← 2 3⍴2 7 1 2 8 2
```



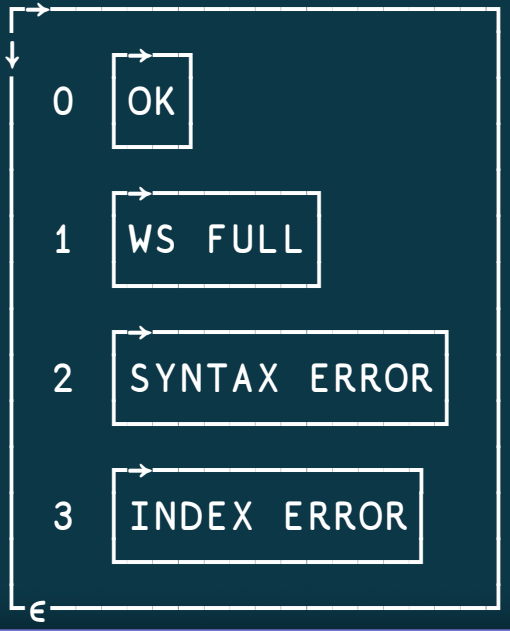
Proposed

```
d←[[3 1 4
   1 5 9]

 [2 7 1
  2 8 2]]
```

Nested table

Current

```
e←⍬⍪0 'OK'
e⍪←1 'WS FULL'
e⍪←2 'SYNTAX ERROR'
e⍪←3 'INDEX ERROR'
e⍪←4 'RANK ERROR'
```



Proposed

```
e←[0 'OK'
   1 'WS FULL'
   2 'SYNTAX ERROR'
   3 'INDEX ERROR'
   4 'RANK ERROR']
```

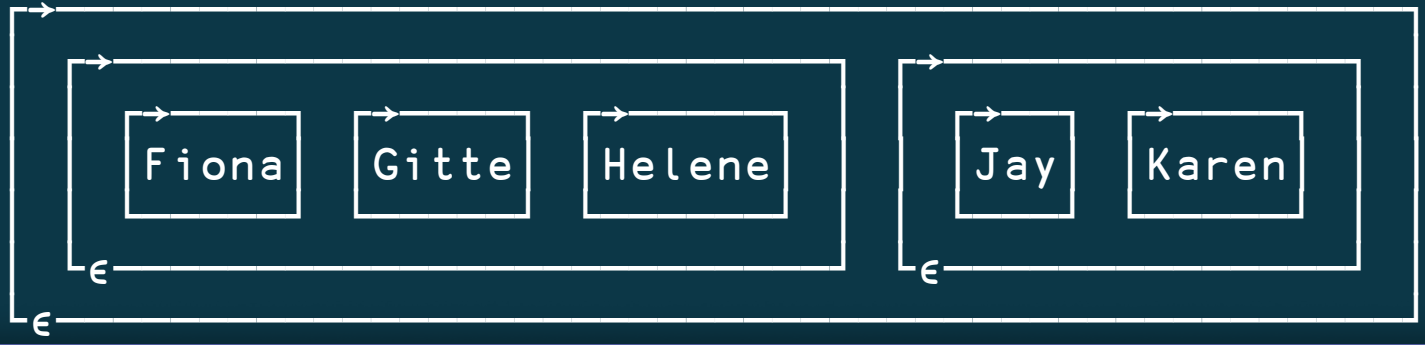
Deeply nested vector

Current

```
l←⊂'Fiona' 'Gitte' 'Helene'
l,←⊂'Jay' 'Karen'
```

Proposed

```
l←(('Fiona'
    'Gitte'
    'Helene')

   ('Jay'
    'Karen'))
```



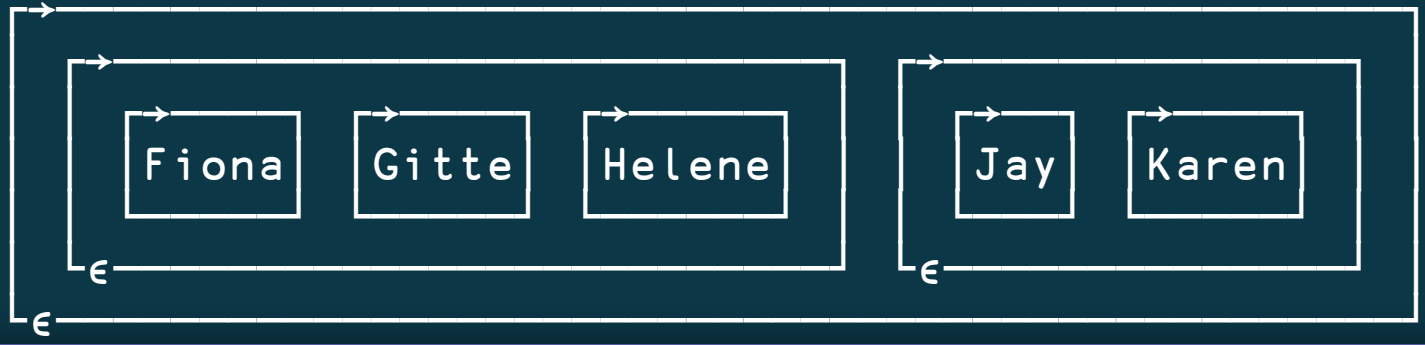
Deeply nested vector

Current

```
l←⊂'Fiona' 'Gitte' 'Helene'
l,←⊂'Jay' 'Karen'
```

Proposed

```
l←('Fiona' 'Gitte' 'Helene'
   'Jay' 'Karen')
```



Deeply nested vector

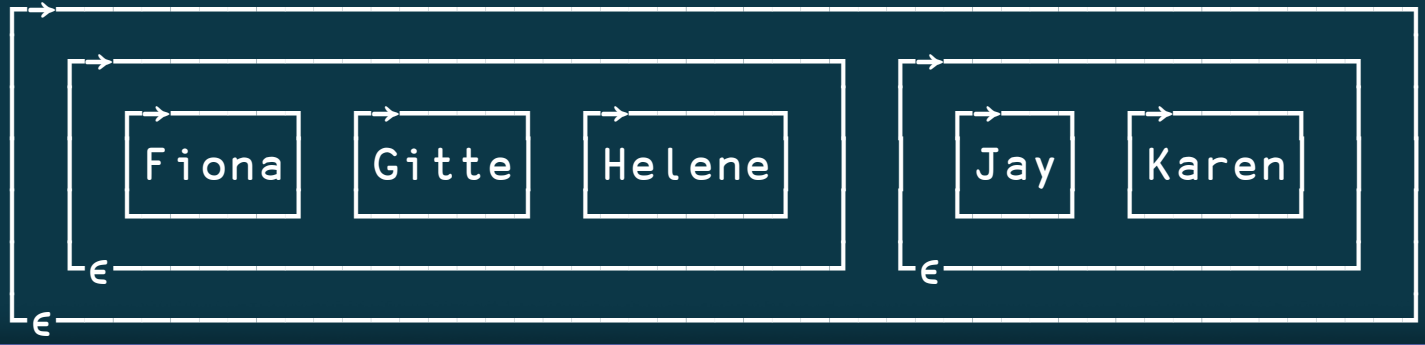
Current

```
l←⊂'Fiona' 'Gitte' 'Helene'
l,←⊂'Jay' 'Karen'
```

Proposed

```
l←('Fiona' 'Gitte' 'Helene'
   'Jay' 'Karen')
```

```
l←('Fiona' 'Gitte' 'Helene' ⋄ 'Jay' 'Karen')
```



(array) assembly

1. The result of each *statement* is collected into a list
2. Any *embedded parentheses* are resolved first; each result becomes an item of the list

```
('Fiona'
 'Gitte'
 'Helene')

(0 'OK'
 1 'WS FULL'
 2 'SYNTAX ERROR'
 3 'INDEX ERROR'
 4 'RANK ERROR')

((3
  1 5 9)
 (2 7 1
  2 8  ))
```

(array) assembly

1. The result of each *statement* is collected into a list
2. Any *embedded parentheses* are resolved first; each result becomes an item of the list

```
('Fiona'
 'Gitte'
 'Helene')

(0 'OK'
 1 'WS FULL'
 2 'SYNTAX ERROR'
 3 'INDEX ERROR'
 4 'RANK ERROR')

((3
  1 5 9)
 (2 7 1
  2 8  ))
```

(array) assembly

1. The result of each *statement* is collected into a list
2. Any *embedded parentheses* are resolved first; each result becomes an item of the list

```
('Fiona'
 'Gitte'
 'Helene')

(0 'OK'
 1 'WS FULL'
 2 'SYNTAX ERROR'
 3 'INDEX ERROR'
 4 'RANK ERROR')

((3
  1 5 9)
 (2 7 1
  2 8  ))
```

[array] assembly

1. The result of each *statement* is collected into a list
2. Any *embedded brackets* are resolved first; each result becomes an item of the list

```
['Fiona'
 'Gitte'
 'Helene']

[0 'OK'
 1 'WS FULL'
 2 'SYNTAX ERROR'
 3 'INDEX ERROR'
 4 'RANK ERROR']

[[3
  1 5 9]
 [2 7 1
  2 8  ]]
```

[array] assembly

1. The result of each *statement* is collected into a list
2. Any *embedded brackets* are resolved first; each result becomes an item of the list

```
['Fiona'
 'Gitte'
 'Helene']

[0 'OK'
 1 'WS FULL'
 2 'SYNTAX ERROR'
 3 'INDEX ERROR'
 4 'RANK ERROR']

[[3
  1 5 9]
 [2 7 1
  2 8  ]]
```

[array] assembly

1. The result of each *statement* is collected into a list
2. Any *embedded brackets* are resolved first; each result becomes an item of the list

```
['Fiona'
 'Gitte'
 'Helene']

[0 'OK'
 1 'WS FULL'
 2 'SYNTAX ERROR'
 3 'INDEX ERROR'
 4 'RANK ERROR']

[[3
  1 5 9]
 [2 7 1
  2 8  ]]
```

[array] assembly

1. The result of each *statement* is collected into a list
2. Any *embedded brackets* are resolved first; each result becomes an item of the list
3. Each item is forced to have minimum rank 1, as if `1⌷⍵` is applied to it.

```
['Fiona'
 'Gitte'
 'Helene']

[0 'OK'
 1 'WS FULL'
 2 'SYNTAX ERROR'
 3 'INDEX ERROR'
 4 'RANK ERROR']

[[3
  1 5 9]
 [2 7 1
  2 8  ]]
```

[array] assembly

1. The result of each *statement* is collected into a list
2. Any *embedded brackets* are resolved first; each result becomes an item of the list
3. Each item is forced to have minimum rank 1, as if `1⌷⍵` is applied to it.
4. *Mix* is applied to the list, producing an array of rank one higher than the highest rank item. I.e. each item of the list becomes a *major cell* of the array which is represented by the nearest surrounding brackets

```
['Fiona'
 'Gitte'
 'Helene']

[0  'OK'
 1  'WS FULL'
 2  'SYNTAX ERROR'
 3  'INDEX ERROR'
 4  'RANK ERROR']

[[3 ⍬
  1 5 9]
 [2 7 1
  2 8 ⍬]]
```

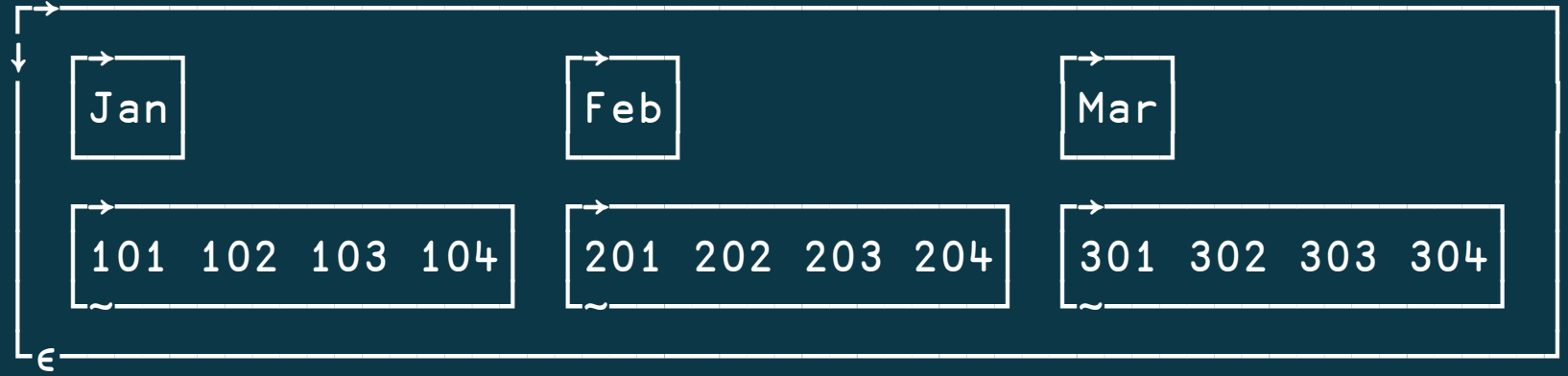
[array] assembly

1. The result of each *statement* is collected into a list
2. Any *embedded brackets* are resolved first; each result becomes an item of the list
3. Each item is forced to have minimum rank 1, as if `1⌷⍵` is applied to it.
4. *Mix* is applied to the list, producing an array of rank one higher than the highest rank item. I.e. each item of the list becomes a *major cell* of the array which is represented by the nearest surrounding brackets

```
['Fiona'
 'Gitte'
 'Helene']

[0  'OK'
 1  'WS FULL'
 2  'SYNTAX ERROR'
 3  'INDEX ERROR'
 4  'RANK ERROR']

[[3 0 0
  1 5 9]
 [2 7 1
  2 8 0]]
```

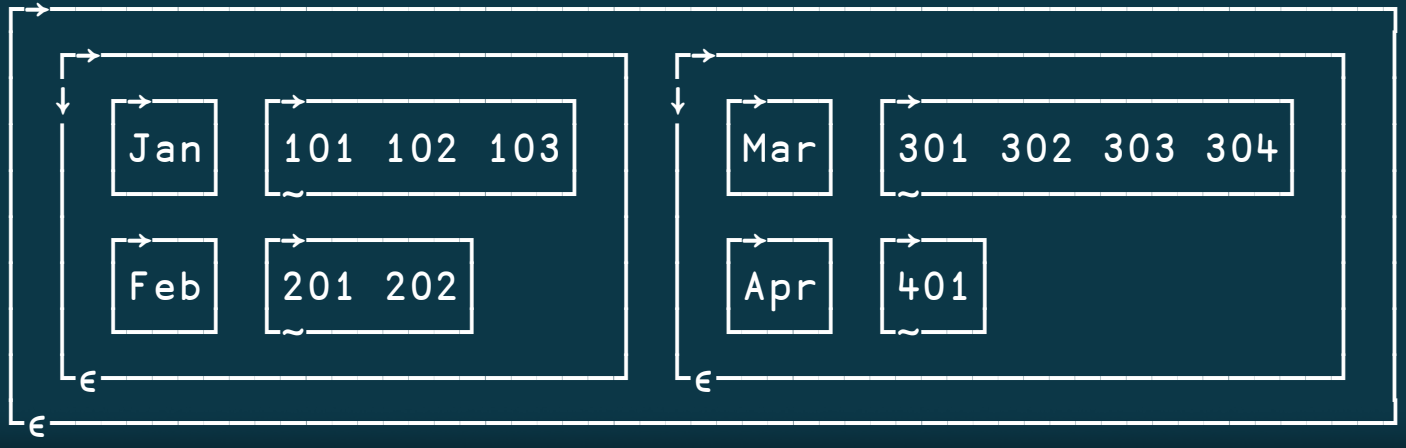
More examples

```
['Jan'                'Feb'                'Mar'
 (101 102 103 104) (201 202 203 204) (301 302 303 304)]
```



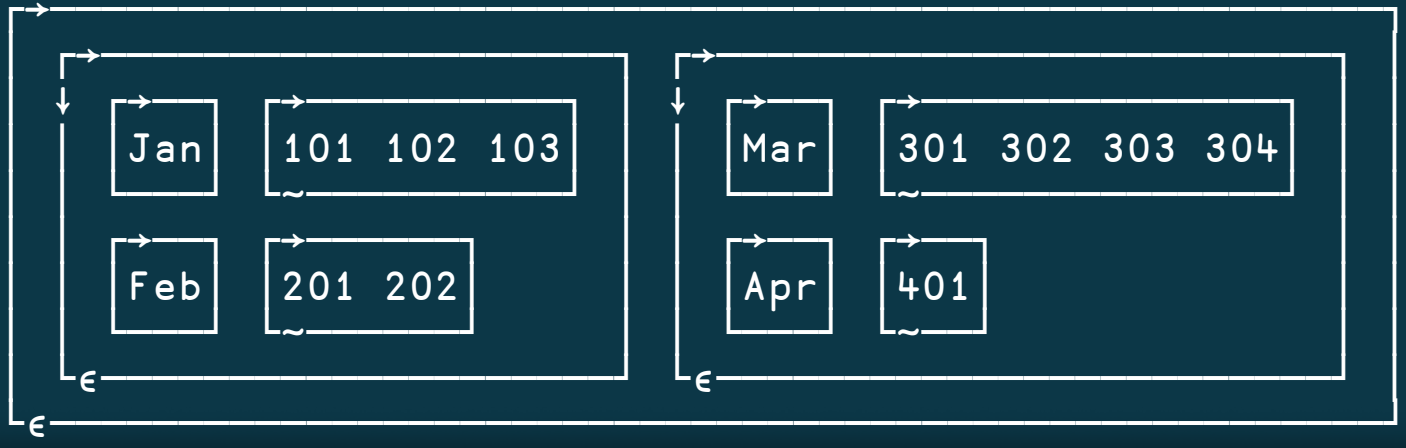
More examples

```
(
 ['Jan' (101 102 103)
  'Feb' (201 202)]

 ['Mar' (301 302 303 304)
  'Apr' (401 ⋄ )]
)
```



More examples

```
(
 ['Jan' (101 102 103)
  'Feb' (201 202)]

 ['Mar' (301 302 303 304)
  'Apr' (301 ◇ )]
)
```



DBMenuCB from]Profile

```
poss←1 2⍴'fns'((0 1)(0.7 0)(0.7 0)×size)
poss,←'fnd'((0 1)(0 0)(0 0)×size)
poss,←'lines'((0 0)(0.7 0)(0.7 0)×size)
poss,←'lnd'((0 0)(0 0)(0 0)×size)
```

DBMenuCB from]Profile

```
poss←1 2⍴'fns'   ((0 1)(0.7 0)(0.7 0)×size)
poss⍪←   'fnd'   ((0 1)(0   0)(0   0)×size)
poss⍪←   'lines'((0 0)(0.7 0)(0.7 0)×size)
poss⍪←   'lnd'   ((0 0)(0   0)(0   0)×size)
```

DBMenuCB from]Profile

```
poss←['fns'   ((0 1)(0.7 0)(0.7 0)×size)
      'fnd'   ((0 1)(0   0)(0   0)×size)
      'lines'((0 0)(0.7 0)(0.7 0)×size)
      'lnd'   ((0 0)(0   0)(0   0)×size)]
```

DBMenuCB from]Profile

```
poss←[('fns'    ◇ (0 1 ◇ 0.7 0 ◇ 0.7 0)×size)
      ('fnd'    ◇ (0 1 ◇ 0   0 ◇ 0   0)×size)
      ('lines'  ◇ (0 0 ◇ 0.7 0 ◇ 0.7 0)×size)
      ('lnd'    ◇ (0 0 ◇ 0   0 ◇ 0   0)×size)]
```

cal from dfns.dws

```
Q1←'January' 'February' 'March    '~¨' '    ⍝ 1st quarter month names.
Q2←'April  ' 'May      ' 'June     '~¨' '    ⍝ 2nd   ..       ..     ..
Q3←'July   ' 'August   ' 'September'~¨' '    ⍝ 3rd   ..       ..     ..
Q4←'October' 'November' 'December '~¨' '    ⍝ 4th   ..       ..     ..
months←Q1,Q2,Q3,Q4                          ⍝ month names for year.
```

cal from dfns.dws

```
Q1←'January'  'February'  'March'      ⍝ 1st quarter month names.
Q2←'April'    'May'       'June'       ⍝ 2nd   ..      ..     ..
Q3←'July'     'August'    'September'  ⍝ 3rd   ..      ..     ..
Q4←'October'  'November'  'December'   ⍝ 4th   ..      ..     ..
months←Q1,Q2,Q3,Q4                     ⍝ month names for year.
```

cal from dfns.dws

```
months←(                                    ⍝ month names for year.
  'January' ◇ 'February' ◇ 'March'          ⍝ 1st quarter month names.
  'April'   ◇ 'May'      ◇ 'June'           ⍝ 2nd   ..       ..     ..
  'July'    ◇ 'August'   ◇ 'September'      ⍝ 3rd   ..       ..     ..
  'October' ◇ 'November' ◇ 'December'       ⍝ 4th   ..       ..     ..
)
```

Eigen from math.dws

```
args←'zheev_'assoc↓⍉↑⍬{                ⍝ associate external fn.
      ⍵,⊂' <C1    ' 'V'}{               ⍝ JOBZ
      ⍵,⊂' <C1    ' 'L'}{               ⍝ UPLO
      ⍵,⊂' <I4    'n}{                  ⍝ N
      ⍵,⊂' =F8[]  '(⍴⍉mat)}{            ⍝ A
      ⍵,⊂' <I4    'n}{                  ⍝ LDA
      ⍵,⊂' >F8[]  'n}{                  ⍝ W
      ⍵,⊂' >F8[]  '(¯2+4×n)}{           ⍝ WORK
      ⍵,⊂' <I4    '(¯1+2×n)}{           ⍝ LWORK
      ⍵,⊂' >F8[]  '(¯2+3×n)}{           ⍝ RWORK
      ⍵,⊂' >I4    ' 0}⍬                 ⍝ INFO
```

Eigen from math.dws

```
args←'zheev_'assoc↓⍉↑⍬{                ⍝ associate external fn.
      ⍵,⊂' <C1    ' 'V'}{               ⍝ JOBZ
      ⍵,⊂' <C1    ' 'L'}{               ⍝ UPLO
      ⍵,⊂' <I4    'n}{                  ⍝ N
      ⍵,⊂' =F8[]  '(∊⍉mat)}{            ⍝ A
      ⍵,⊂' <I4    'n}{                  ⍝ LDA
      ⍵,⊂' >F8[]  'n}{                  ⍝ W
      ⍵,⊂' >F8[]  '(¯2+4×n)}{           ⍝ WORK
      ⍵,⊂' <I4    '(¯1+2×n)}{           ⍝ LWORK
      ⍵,⊂' >F8[]  '(¯2+3×n)}{           ⍝ RWORK
      ⍵,⊂' >I4    ' 0}⍬                 ⍝ INFO
```

Eigen from math.dws

```
args←'zheev_'assoc↓⍉↑⍬{                    ⍝ associate external fn.
      ⍵,⊂' <C1 '    'V'          }{        ⍝ JOBZ
      ⍵,⊂' <C1 '    'L'          }{        ⍝ UPLO
      ⍵,⊂' <I4 '     n           }{        ⍝ N
      ⍵,⊂' =F8[] ' (∊⍉mat)       }{        ⍝ A
      ⍵,⊂' <I4 '     n           }{        ⍝ LDA
      ⍵,⊂' >F8[] '   n           }{        ⍝ W
      ⍵,⊂' >F8[] ' (¯2+4×n)      }{        ⍝ WORK
      ⍵,⊂' <I4 '    (¯1+2×n)     }{        ⍝ LWORK
      ⍵,⊂' >F8[] ' (¯2+3×n)      }{        ⍝ RWORK
      ⍵,⊂' >I4 '     0           }⍬        ⍝ INFO
```

Eigen from math.dws

```
args←'zheev_'assoc↓⍉↑(                  ⍝ associate external fn.
        ' <C1 '    'V'                      ⍝ JOBZ
        ' <C1 '    'L'                      ⍝ UPLO
        ' <I4 '     n                       ⍝ N
        ' =F8[] ' (∊⍉mat)                   ⍝ A
        ' <I4 '     n                       ⍝ LDA
        ' >F8[] '   n                       ⍝ W
        ' >F8[] ' (¯2+4×n)                  ⍝ WORK
        ' <I4 '    (¯1+2×n)                 ⍝ LWORK
        ' >F8[] ' (¯2+3×n)                  ⍝ RWORK
        ' >I4 '     0          )            ⍝ INFO
```

Eigen from math.dws

```
args←'zheev_'assoc↓⍉[                  ⍝ associate external fn.
        ' <C1 '    'V'                      ⍝ JOBZ
        ' <C1 '    'L'                      ⍝ UPLO
        ' <I4 '     n                       ⍝ N
        ' =F8[] ' (,⍉mat)                  ⍝ A
        ' <I4 '     n                       ⍝ LDA
        ' >F8[] '   n                       ⍝ W
        ' >F8[] ' (¯2+4×n)                  ⍝ WORK
        ' <I4 '    (¯1+2×n)                 ⍝ LWORK
        ' >F8[] ' (¯2+3×n)                  ⍝ RWORK
        ' >I4 '     0                ]      ⍝ INFO
```

morse from dfns.dws

```
morse←{                                 ⍝ Conversion to/from Morse code.
    P M←{⍵~¨' '}⍉↓⍪⍉↑{                  ⍝ plain-text and Morse codes.
        ('A' '.-'   )('B' '-...' )('C' '-.-.' )('D' '-..'  ),⍵}{
        ('E' '.'    )('F' '..-.' )('G' '--.'  )('H' '....' ),⍵}{
        ('I' '..'   )('J' '.---' )('K' '-.-'  )('L' '.-..' ),⍵}{
        ('M' '--'   )('N' '-.'   )('O' '---'  )('P' '.--.' ),⍵}{
        ('Q' '--.-' )('R' '.-.'  )('S' '...'  )('T' '-'    ),⍵}{
        ('U' '..-'  )('V' '...-' )('W' '.--'  )('X' '-..-' ),⍵}{
        ('Y' '-.--' )('Z' '--..' ),⍵}{

        ('0' '-----')('1' '.----')('2' '..---')('3' '...--'),⍵}{
        ('4' '....-')('5' '.....')('6' '-....')('7' '--...'),⍵}{
        ('8' '---..')('9' '----.'),⍵}{

        ('.' '.-.-.-')(',' '--..--')(':' '---...'),⍵}{
        ('?' '..--..')('''' '.----.')('-' '-....-'),⍵}{
        ('/' '-..-.' )('(' '-.--.-')(')' '-.--.-'),⍵}{
        ('"' '.-..-.')('@' '.--.-.')('=' '-...-' ),⍵}{

        ⍵}⊂' ' ' / '                    ⍝ A blank / inter-word separator.

    1=|≡,⍵:M[P⍳⍵∩P]                     ⍝ A plain text to Morse.
    2=|≡,⍵:P[M⍳⍵∩M]                     ⍝ A Morse to plain text.
}
```

morse from dfns.dws

```
morse←{                          ⍝ Conversion to/from Morse code.
    P M←{⍵~¨' '}⎕CY'dfns'{       ⍝ plain-text and Morse codes.
        ('A' '.-'  )('B' '-...' )('C' '-.-.' )('D' '-..'  ),⍵}{
        ('E' '.'   )('F' '..-.' )('G' '--.'  )('H' '....' ),⍵}{
        ('I' '..'  )('J' '.---' )('K' '-.-'  )('L' '.-..' ),⍵}{
        ('M' '--'  )('N' '-.'   )('O' '---'  )('P' '.--.' ),⍵}{
        ('Q' '--.-')('R' '.-.'  )('S' '...'  )('T' '-'    ),⍵}{
        ('U' '..-' )('V' '...-' )('W' '.--'  )('X' '-..-' ),⍵}{
        ('Y' '-.--')('Z' '--..' ),⍵}{

        ('0' '-----')('1' '.----')('2' '..---')('3' '...--'),⍵}{
        ('4' '....-')('5' '.....')('6' '-....')('7' '--...'),⍵}{
        ('8' '---..')('9' '----.'),⍵}{

        ('.' '.-.-.-')(',' '--..--')(':' '---...'),⍵}{
        ('?' '..--..')('¡' '.----.')('-' '-....-'),⍵}{
        ('/' '-..-.' )('(' '-.--.-')(')' '-.--.-'),⍵}{
        ('"' '.-..-.')('@' '.--.-.')('=' '-...-' ),⍵}{

        ⍵}⊂' ' ' / '             ⍝ blank / inter-word separator.

    1=|≡,⍵:M[P⍳⍵∩P]              ⍝ plain text to Morse.
    2=|≡,⍵:P[M⍳⍵∩M]              ⍝ Morse to plain text.
}
```

morse from dfns.dws

```
morse←{                          ⍝ Conversion to/from Morse code.
    P M←↓⍉↑(                     ⍝ plain-text and Morse codes.
          'A' '.-'   ◇ 'B' '-...' ◇ 'C' '-.-.' ◇ 'D' '-..'
          'E' '.'    ◇ 'F' '..-.' ◇ 'G' '--.'  ◇ 'H' '....'
          'I' '..'   ◇ 'J' '.---' ◇ 'K' '-.-'  ◇ 'L' '.-..'
          'M' '--'   ◇ 'N' '-.'   ◇ 'O' '---'  ◇ 'P' '.--.'
          'Q' '--.-' ◇ 'R' '.-.'  ◇ 'S' '...'  ◇ 'T' '-'
          'U' '..-'  ◇ 'V' '...-' ◇ 'W' '.--'  ◇ 'X' '-..-'
          'Y' '-.--' ◇ 'Z' '--..'

          '0' '-----' ◇ '1' '.----' ◇ '2' '..---' ◇ '3' '...--'
          '4' '....-' ◇ '5' '.....' ◇ '6' '-....' ◇ '7' '--...'
          '8' '---..' ◇ '9' '----.'

          '.' '.-.-.-' ◇ ',' '--..--' ◇ ':' '---...'
          '?' '..--..' ◇ ';' '-.-.-.' ◇ '-' '-....-'
          '/' '-..-.'  ◇ '(' '-.--.-' ◇ ')' '-.--.-'
          '"' '.-..-.' ◇ '@' '.--.-.' ◇ '=' '-...-'

          ' ' ' / ' ) ⍝ A blank / inter-word separator.

    1=⌊|≡,⍵:M[P⍳⍵∩P]             ⍝ A plain text to Morse.
    2=⌊|≡,⍵:P[M⍳⍵∩M]             ⍝ A Morse to plain text.
}
```

morse from dfns.dws

```
morse←{
    P M←↓⍉↑(
          'A' '.-'   ◇ 'B' '-...' ◇ 'C' '-.-.' ◇ 'D' '-..'
          'E' '.'    ◇ 'F' '..-.' ◇ 'G' '--.'  ◇ 'H' '....'
          'I' '..'   ◇ 'J' '.---' ◇ 'K' '-.-'  ◇ 'L' '.-..'
          'M' '--'   ◇ 'N' '-.'   ◇ 'O' '---'  ◇ 'P' '.--.'
          'Q' '--.-' ◇ 'R' '.-.'  ◇ 'S' '...'  ◇ 'T' '-'
          'U' '..-'  ◇ 'V' '...-' ◇ 'W' '.--'  ◇ 'X' '-..-'
          'Y' '-.--' ◇ 'Z' '--..'

          '0' '-----' ◇ '1' '.----' ◇ '2' '..---' ◇ '3' '...--'
          '4' '....-' ◇ '5' '.....' ◇ '6' '-....' ◇ '7' '--...'
          '8' '---..' ◇ '9' '----.'

          '.' '.-.-.-' ◇ ',' '--..--' ◇ ':' '---...'
          '?' '..--..' ◇ ';' '-.-.-.' ◇ '-' '-....-'
          '/' '-..-.'  ◇ '(' '-.--.-' ◇ ')' '-.--.-'
          '"' '.-..-.' ◇ '@' '.--.-.' ◇ '=' '-...-'

          ' ' ' / ' ) ⍝ A blank / inter-word separator.

    1=|≡,⍵:M[P⍳⍵∩P] ⍝ A plain text to Morse.
    2=|≡,⍵:P[M⍳⍵∩M] ⍝ A Morse to plain text.
}
```

Summary: Array Notation

- This notation is to APL what JSON arrays are to JavaScript et al.
  e.g. to learn APL arrays without learning `⍴ ⊂ ,` first
- Makes APL arrays read/write accessible to others:
  e.g. APL, J, MATLAB, Python's NumPy
- Use any text editor to edit (practical at least for simple cases):
  variables
  constants in tacit functions
  `(2 2⍴2 7 1 8)∘.+⊢` becomes `[2 7 ⋄ 1 8]∘.+⊢`
- Save constant data as plain-text:
  e.g. for SCM (GitHub, et al.), collaborative editing, 3rd party editors
- Comments inside code for arrays

Why not just use JSON?

Why not just use JSON?

1. No concept of rank (only depth)

   APL `⍪0` ⇒ JSON `[[0]]` ⇒ APL `,⊂,0`

Why not just use JSON?

1. No concept of rank (only depth)

   APL `⍪0` ⇒ JSON `[[0]]` ⇒ APL `,⊂,0`

2. No concept of scalar characters

   APL `'a'0` ⇒ JSON `["a",0]` ⇒ APL `(,'a')0`

Why not just use JSON?

1. No concept of rank (only depth)

   APL `⍪0` ⇒ JSON `[[0]]` ⇒ APL `,⊂,0`

2. No concept of scalar characters

   APL `'a'0` ⇒ JSON `["a",0]` ⇒ APL `(,'a')0`

3. Notation clashes with APL

   e.g. JSON vectors: `[]` `[1]` `[1,2]`

Why not just use JSON?

1. No concept of rank (only depth)
   APL `⍪0` ⇒ JSON `[[0]]` ⇒ APL `,⊂,0`
2. No concept of scalar characters
   APL `'a'0` ⇒ JSON `["a",0]` ⇒ APL `(,'a')0`
3. Notation clashes with APL
   e.g. JSON vectors: `[]` `[1]` `[1,2]`
4. So one cannot use APL expressions inline:
   e.g what is `'abc'[1,2]`?

Why not just use JSON?

1. No concept of rank (only depth)
   APL `⍪0` ⇒ JSON `[[0]]` ⇒ APL `,⊂,0`
2. No concept of scalar characters
   APL `'a'0` ⇒ JSON `["a",0]` ⇒ APL `(,'a')0`
3. Notation clashes with APL
   e.g. JSON vectors: `[]` `[1]` `[1,2]`
4. So one cannot use APL expressions inline:
   e.g what is `'abc'[1,2]`? `'ab'`

Why not just use JSON?

1. No concept of rank (only depth)
   APL `⍪0` ⇒ JSON `[[0]]` ⇒ APL `,⊂,0`
2. No concept of scalar characters
   APL `'a'0` ⇒ JSON `["a",0]` ⇒ APL `(,'a')0`
3. Notation clashes with APL
   e.g. JSON vectors: `[]` `[1]` `[1,2]`
4. So one cannot use APL expressions inline:
   e.g what is `'abc'[1,2]`? `'abc'(1 2)`

Why not just use JSON?

1. No concept of rank (only depth)
   APL `⍪0` ⇒ JSON `[[0]]` ⇒ APL `,⊂,0`
2. No concept of scalar characters
   APL `'a'0` ⇒ JSON `["a",0]` ⇒ APL `(,'a')0`
3. Notation clashes with APL
   e.g. JSON vectors: `[]` `[1]` `[1,2]`
4. So one cannot use APL expressions inline:
   e.g what is `'abc'[1,2]`? `'abc'(,⊂1 2)`

Why not just use JSON?

Edge cases: Rank

Single-column matrices

```
[1
 2      ⇔      [1 ⋄ 2 ⋄ 3]
 3]
```

Single-row matrices

```
[
 1 2 3      ⇔      [1 2 3 ⋄ ]
     ]
```

Trailing length-zero axis

```
[0
 0      ⇔      [0 ⋄ 0 ⋄ 0]
 0]
```

Edge cases: Depth

Simple vectors written vertically

```
(1
 2      ⇔   [1 ⋄ 2 ⋄ 3]
 3)
```

Single-element nested vectors

```
(
 1 2 3    ⇔    (1 2 3 ⋄ )
      )
```

Vectors of vectors

```
(1 2          (1 2
 ,3      ⇔     (3⋄)
 4 5)          4 5)
```

Shortcomings

Enclosed scalars

```
⊂2 2⍴2 7 1 8

⊂[2 7 ◇ 1 8]
```

Non-trailing length-zero axes

```
0 3⍴0

0≠[
    0 0 0
        ]
```

```
2 2⍴⊂¨(1 2)(2 3)(2 3)(3 4)

[(⊂1 2)(⊂2 3)
 (⊂2 3)(⊂3 4)]
```

Empty non-simple

```
0⍴⊂0 0
```

Awkwardnesses

High-rank elements

```
2  2⍴⊂2  2⍴2  7  1  8
```

Awkwardnesses

High-rank elements

```
2  2⍴⊂2  2⍴2  7  1  8

  [[2  7   [2  7
    1  8]    1  8]

   [2  7   [2  7
    1  8]    1  8]]
```

←WRONG!

Awkwardnesses

High-rank elements

```
2 2⍴⊂2 2⍴2 7 1 8

[[2 7 [2 7
      1 8] 1 8]

  [2 7  [2 7
   1 8]  1 8]]
```

←WRONG!

Awkwardnesses

High-rank elements

```
2  2⍴⊂2  2⍴2  7  1  8
```

Awkwardnesses

High-rank elements

```
2 2⍴⊂2 2⍴2 7 1 8

┌→──┬→──┐
↓2 7│2 7│
│1 8│1 8│
├→──┼→──┤
↓2 7│2 7│
│1 8│1 8│
└───┴───┘
```

Awkwardnesses

High-rank elements

```
2 2⍴⊂2 2⍴2 7 1 8

  [[2 7
    1 8] [2 7
          1 8]
   [2 7
    1 8] [2 7
          1 8]]

[[2 7 ◇ 1 8] [2 7 ◇ 1 8]
 [2 7 ◇ 1 8] [2 7 ◇ 1 8]]
```

Summary: Arrays

Rank

```
[1 2
 3 3  ⇔ [1 2 ⋄ 3 4 ⋄ 5 6]
 4 5]
```

```
[1
 2   ⇔ [1 ⋄ 2 ⋄ 3]
 3]
```

Depth

```
(1 2
 3 4  ⇔ (1 2 ⋄ 3 4 ⋄ 5 6)
 5 6)
```

```
(1
 2   ⇔ (1 ⋄ 2 ⋄ 3)
 3)
```

DYALOG

Belfast 2018

```
(Title:['Array'
        'Notation'
        'Mk III']

 Presenter:('Adám'
            'Brudzewsky'))
```

adam@dyalog.com

Namespace Notation

We don't have any such notation!

We can only create namespaces through:

- side-effects of otherwise unrelated actions
- edge-case usage of system functions
- converting from JSON

Why?

- This notation is to APL what JSON objects are to JavaScript et al.
  e.g. to learn basic APL OO without learning ⎕NS ⎕FIX ⎕JSON first
- Makes APL namespaces read/write accessible to others:
  e.g. APL, J, MATLAB, Python's NumPy
- Use any text editor to edit unscripted namespaces
- Save unscripted namespaces as plain text:
  e.g. for SCM (GitHub, et al.), collaborative editing, 3rd party editors
- Comments inside namespace-creating code without being part of script

continues…

Why?

… continued

- Temporary namespace to supply evaluation context and named arguments/operands, including functions:

```
Model(steps:100 ⋄ f:×⍨-⊢ ⋄ file:'/tmp/out.txt')

⎕CSV⍠(Decimal:',' ⋄ Trim:0)

'1st' '2nd'⎕R(Trans1:'0th' ⋄ Trans2:{⍵.Match})

(⎕CT:0).=
(⎕DIV:1).÷
(⎕IO:0).(⍳∘≠≡⍋⊃⍨⊂)
```

Namespace Notation

Current

```
ns←⎕NS ⍬
ns.life←42
ns.lang←'APL'
```

Proposed

```
ns←(life:42
    lang:'APL')
```

JSON

```
{"life":42,
 "lang":"APL"}
```

Inline

Current `(⎕NS ⍬).(life lang)←42 'APL'`

Proposed `(life:42 ⋄ lang:'APL')`

JSON `{"life":42, "lang":"APL"}`

Inline

Current

```
(⎕NS ⍬).(life lang)←42 'APL'
{α←⎕NS ⍬ ⋄ α.(life lang)←⍵ ⋄ α}42 'APL'
```

Proposed

```
(life:42 ⋄ lang:'APL')
```

JSON

```
{"life":42, "lang":"APL"}
```

Functions

Current

```
ns←⎕NS ⍬
ns.dfn←{
            ⍺+⍵
        }
∇r←a Tradfn b
 r←a+b
∇
'ns'⎕NS'Tradfn'
⎕EX'Tradfn'
```

Proposed

```
ns←(dfn:{
            ⍺+⍵
        }
    ∇r←a Tradfn b
     r←a+b
    ∇
)
```

Possible: Scripts

Current

```
ns←⎕NS ⍬
ns.⎕FIX ':Class C' '  :Field f' ':EndClass'
ns.⎕FIX ':Namespace Ns' '  var←42' ':EndNamespace'
```

Proposed

```
ns←(:Class C
      :Field f
    :EndClass
    :Namespace Ns
      var←42
    :EndNamespace
   )
```

Current

Unscripted Namespace (containing scripted namespaces)

```
ns←⎕NS ⍬
ns.⎕FIX ':Class C' '  :Field f' ':EndClass'
ns.⎕FIX ':Namespace Ns' '  var←42' ':EndNamespace'
```

Scripted Namespace (containing scripted namespaces)

```
src←⊂':Namespace'
src,←':Class C' '  :Field f' ':EndClass'
src,←':Namespace Ns' '  var←42' ':EndNamespace'
src,←⊂':EndNamespace'
ns←⎕FIX src
⎕EX'src'
```

Possible

Unscripted Namespace
(containing scripted namespaces)

```
ns←(
     :Class C
       :Field f
     :EndClass

     :Namespace Ns
       var←42
     :EndClass
   )
```

Proposed

Scripted Namespace
(containing scripted namespaces)

```
ns←⎕FIX(':Namespace'
        '   :Class C'
        '     :Field f'
        '   :EndClass'
        ''
        '   :Namespace Ns'
        '     var←42'
        '   :EndClass'
        ':EndNamespace')
```

Mixed Bag Example

```
utils←(
    ∇ res←avg nums;count   ⍝ tradfn
      total←+/nums
      count←≢nums
      res←total÷count
    ∇
    identity3:[1 0 0   ⍝ matrix
               0 1 0
               0 0 1]
    product:('Dyalog'   ⍝ "VTV"
             'APL')
    Link:{(⊂⍺),⊆⍵}   ⍝ dfn
    Split:≠⊆⊢   ⍝ train
    primes:(⊢~∘.×⍨)1↓⍳100   ⍝ expression
)
```

Game of Life Patterns

Current

```
pats←⎕NS ⍬
pats.Glider←⍉⍪0 0 1
pats.Glider,←⍪1 0 1
pats.Glider,←⍪0 1 1

pats.RPentomino←⍉⍪0 1 1
pats.RPentomino,←⍪1 1 0
pats.RPentomino,←⍪0 1 0

pats.BiStable←⍉⍪0 1 1 1
pats.BiStable,←⍪1 1 1 0

pats.Stable←⍉⍪0 1 1 0
pats.Stable,←⍪1 0 0 1
pats.Stable,←⍪0 1 1 0
```

Proposed

```
pats←(
    Glider:[0 0 1
            1 0 1
            0 1 1]

    RPentomino:[0 1 1
                1 1 0
                0 1 0]

    BiStable:[0 1 1 1
              1 1 1 0]

    Stable:[0 1 1 0
            1 0 0 1
            0 1 1 0])
```

Empty Namespace

Current

```
⎕NS ⍬
```

Proposed

```
( )
```

JSON

```
{}
```

Scope

```
a←1
r←(
    a:2
    b:a←3
    c:a
  )
```

Scope

```
a←1
r←(
    a:2
    b:a←3
    c:a
  )
r.a
```

Scope

```
      a←1
      r←(
          a:2
          b:a←3
          c:a
        )
      r.a
2
```

Scope

```
      a←1
      r←(
          a:2
          b:a←3
          c:a
        )
      r.a
2
      r.c
```

Scope

```
      a←1
      r←(
          a:2
          b:a←3
          c:a
        )
      r.a
2
      r.c
1
```

Scope

```
a←1
r←(
    a:2
    b:a←3
    c:a
  )
r.a
```

```
2
```

```
r.c
```

```
1
```

```
a←1
r←(
    a:##.{2}⍬
    b:##.{a←3}⍬
    c:##.{a}⍬
  )
r.a
```

```
2
```

```
r.c
```

```
1
```

Scope

```
      a←1
      r←(
          a:2
          b:a←3
          c:a
      )
      r.a
2

      r.c
1
```

```
a←1
:Namespace r
    a←##.{2}⍬
    b←##.{a←3}⍬
    c←##.{a}⍬
:EndNamespace
r.a
2

r.c
1
```

Bonus: Populating Namespaces

Bonus: Populating Namespaces

```
myns←{α←⎕NS ⍬ ⋄ α.(life lang)←⍵ ⋄ α}42 'APL'
```

Bonus: Populating Namespaces

```
myns←{α←⎕NS ⍬ ⋄ α.(life lang)←⍵ ⋄ α}42 'APL'
```

```
names←'life' 'lang'
vals←42 'APL'
```

Bonus: Populating Namespaces

```
myns←{α←⎕NS ⍬ ⋄ α.(life lang)←⍵ ⋄ α}42 'APL'
```

```
names←'life' 'lang'
vals←42 'APL'
```

```
myns←names {tmp←⎕NS ⍬ ⋄ α tmp.{⍎⍺,'←⍵'}¨⍵ ⋄ ns} vals
```

Bonus: Populating Namespaces

```
myns←{α←⎕NS ⍬ ⋄ α.(life lang)←⍵ ⋄ α}42 'APL'
```

```
names←'life' 'lang'
vals←42 'APL'
```

```
myns←names {tmp←⎕NS ⍬ ⋄ α tmp.{⍎⍺,'←⍵'}¨⍵ ⋄ ns} vals
```

```
myns←⎕NS names vals
```

Bonus: Populating Namespaces

```
myns←{α←⎕NS ⍬ ⋄ α.(life lang)←⍵ ⋄ α}42 'APL'

names←'life' 'lang'
vals←42 'APL'

myns←names {tmp←⎕NS ⍬ ⋄ α tmp.{⍎⍺,'←⍵'}¨⍵ ⋄ ns} vals

myns←⎕NS names vals

(names vals)←(⎕NS⍠¯1) myns
```

Bonus: Populating Namespaces

```
myns←{α←⎕NS ⍬ ⋄ α.(life lang)←⍵ ⋄ α}42 'APL'

names←'life' 'lang'
vals←42 'APL'

myns←names {tmp←⎕NS ⍬ ⋄ α tmp.{⍎⍺,'←⍵'}¨⍵ ⋄ ns} vals

myns←⎕NS names vals

(names vals)←(⎕NS⍠¯1) myns
```

Summary: Namespaces

Literal

```
(life:42 ⋄ lang:'APL')
```

Empty

```
()
```

Populate

```
⎕NS ('life' 'lang') (42 'World')
```

DYALOG

Belfast 2018

```
(Title:['Array      '
        'Notation'
        'Mk III   ']

 Presenter:('Adám'
            'Brudzewsky'))
```

adam@dyalog.com