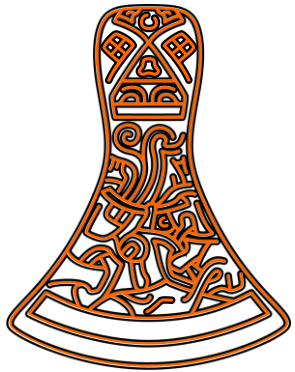
DYALOG

Glasgow 2024

Selected Primitives



Brian Becker

Peter Mikkelsen

Silas Poulson

Welcome!

- Introductions
  - Name
  - How much APL experience?
  - Goals
- Agenda

  60 15 60 15 60
- Participate
  - Ask questions
  - Work together
  - Ask for help if you need it
- Have fun
- `(⎕IO ⎕ML)←1`

Tally

≠Y

Just an average APL expression

```
      avg ← +/÷⍴   ⍝ {(+/⍵)÷⍴⍵}
      avg 3 1 4 1 5 9 2 6
3.875
      avg 42

      avg 3 4⍴⍳12
LENGTH ERROR
```

Just an average APL expression

```
      avg ← +⌿÷≢   ⍝ {(+⌿⍵)÷≢⍵}
      avg 3 1 4 1 5 9 2 6
3.875
      avg 42
42
      avg 3 4⍴⍳12
5 6 7 8
```

Tally versus Shape

```
      ≠'Mississippi'
11
      ⍴'Mississippi'
11
      ]display ≠'Mississippi'
11
      ]display ⍴'Mississippi'
┌→┐
│11│
└~┘
```

Tally versus Shape

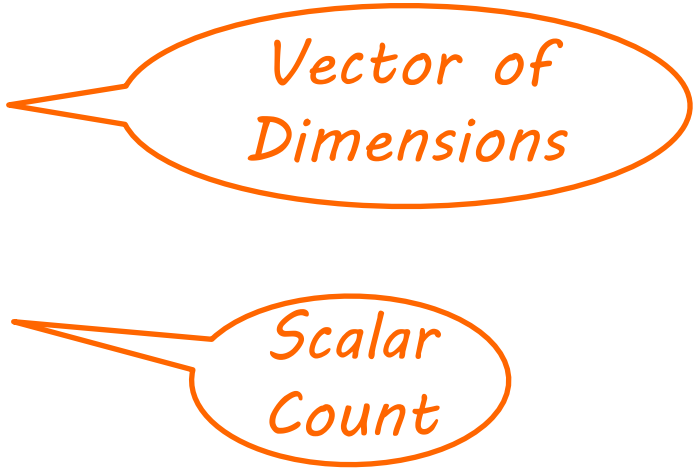
```
      ⊢scalar←42
42

      ⍴scalar
```


```
      ≢scalar
```


```
1
```

Tally versus Shape

```
      ⊢mat←3 4⍴⍳12
1  2  3  4
5  6  7  8
9 10 11 12
      ⍴mat
3 4

      ≢mat
3
```



Tally: Leading Axis

```
      ≢2 3⍴⎕A
2
      ⍴2 3⍴⎕A
2 3
      ≢4 5 6 7 8 ⍴ ⍳6720
4
```

Tally: Exercises

1. Given a matrix and an array which may be either a vector or a matrix, write a function to conditionally concatenate the array to the matrix if their shapes are conforming.

```
    (2 3⍴⍳6){fn}'ab'          (2 3⍴⍳6){fn}'abc'
1 2 3 a                   1 2 3
4 5 6 b                   4 5 6
```

2. Write an expression to test if an array A is empty.

Tally: Exercises Solutions

Note: The solutions presented in this workshop are not the only possible solutions to the exercises

1.
```
{α=⍤≠ω: α,ω ◇ α}
{(≠α)=≠ω: α,ω ◇ α}
```

2.
```
0∈⍴A
```
Why isn't ≠ used? Because it may be that a non-leading dimension is 0.
```
      A←3 0 4⍴42
      0=≠A
0
      0∈⍴A
1
```

Unique Mask

Unique Mask

```
      ∪'Mississippi'
Misp
      ≠'Mississippi'
1 1 1 0 0 0 0 0 1 0 0

      ↑'Mississippi' (≠'Mississippi')
M i s s i s s i p p i
1 1 1 0 0 0 0 0 1 0 0

      (≠'Mississippi') ≠ 'Mississippi'
Misp
```

Unique Mask

```
      (≠'Mississippi') ≠ 'Mississippi'
Misp
      (≠≠⊢)'Mississippi'
1
```

We'll come back to this soon

Unique Mask – Leading Axis

```
      ≠m
1 1 0 0

      (≠m) ⌿ m
 1  2  3
 4  5  6

 7  8  9
10 11 12
```

```
      ⊢m←(4 2 3⍴⍳12)
 1  2  3
 4  5  6

 7  8  9
10 11 12

 1  2  3
 4  5  6

 7  8  9
10 11 12
```

Duplicate Elements

```
      ≠ 'Mississippi'
1 1 1 0 0 0 0 0 1 0 0

      ~≠ 'Mississippi'
0 0 0 1 1 1 1 1 0 1 1

      (~≠'Mississippi') ≠ 'Mississippi'
sissipi
```

Tangent -]runtime

```
mat ← ? 1000 1000 ⍴ 1000000

]runtime -c +/,mat +/∊mat +/+/mat +⌿+⌿mat

+/,mat   → 5.3E¯4 |    0% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
+/∊mat   → 5.2E¯4 |   -1% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
+/+/mat  → 2.0E¯4 | -64% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
+⌿+⌿mat  → 6.5E¯4 | +22% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
```

Count Uniques

```
      d← ? 1e6 ⍴ 1e6

      (≢∪d) (+/≠d)
632453 632453

      ]runtime -c ≢∪d +/≠d

  ≢∪d  → 9.7E¯3 |  0% ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
  +/≠d → 1.0E¯2 | +7% ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
```

Unique Mask Exercises

1. Oh no! Your ∪ key is broken, and there's no language bar available, and you can't copy and paste... Write a function to return the unique duplicated letters in a word.

```
      {your-function}'Mississippi'
sip
```

Unique Mask Exercise Solutions

1. `(≠⊢⍤0 1⊢)(~∘≠⊢⍤0 1⊢)`
   `{{⍵/⍨≠⍵}⍵/⍨~≠⍵}`

Left/Right/Same

```
X ⊣ Y
X ⊢ Y
  ⊣ Y
  ⊢ Y
```

Left/Right/Same

```
      23 ⊣ 42 ⍝ left
23
      23 ⊢ 42 ⍝ right
42
         ⊣ 42 ⍝ same
42
         ⊢ 42 ⍝ same
42
```

Left/Right/Same

	⊣	⊢
Monadic	`{⍵}`	`{⍵}`
Dyadic	`{⍺}`	`{⍵}`

Ambivalent	`{⍺←⍵ ⋄ ⍺}`
	`{⍵}⍨`
	`⊢⍨`
		`{⍵}`
		`{⍺}⍨`
		`⊣⍨`

The "Anti-diamond"

```
(rc msg)←¯1 'oops' ⋄ →0

→0 ⊣ (rc msg)←¯1 'oops'
```

The "Anti-diamond"

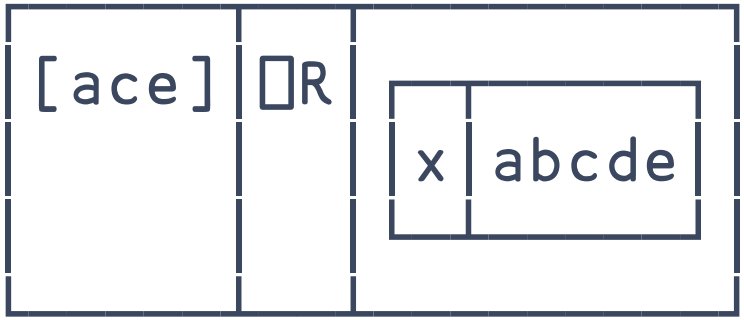
```
      'myapp.log' Log 'Test failed!'
LOGGED: Test failed!

      Log←{⍵ ⎕NPUT ⍺ 2 ⋄ LOGGED: ',⍵}

      Log←{'LOGGED: ',⍵ ⊣ ⍵ ⎕NPUT ⍺ 2}
```

Replace one result with another

```
      Append←{ tn←⍺ ⎕NTIE 0
               (⎕NUNTIE tn) ⊢ ⍵ ⎕NAPPEND tn }

      '/tmp/foo.msgs' Append 'my message'
10
```

Separate right operand and argument

```
      '[ace]' ⍠R 'x' 'abcde'
```




```
      '[ace]' ⍠R 'x'⊢'abcde'
xbxdx
```

Separate right operand and argument

```
      '[ace]' ⌷R 'x'⊢'abcde'
xbxdx

      ('[ace]' ⌷R 'x')'abcde'
xbxdx
```

Address arguments in tacit functions

⊢ refers to the right argument

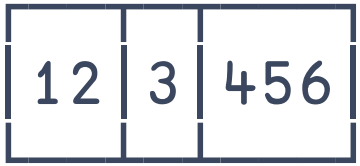
```
      (⊂∘⍋⌷⊢) 'sort me'
 emorst
```

⊣ refers to the left argument

```
      'split-me-on' (≠⊆⊣) '-'
```

Address arguments in tacit functions

```
'12,3,456' {(⍺≠⍵)⊆⍺} ','
```



Address arguments in tacit functions

```
{(⍺≠⍵)⊆⍺} ⍞ replace {} with ()
```

Address arguments in tacit functions

```
((α≠ω)⊆α) ⍘ (α f ω) → f
```

Address arguments in tacit functions

```
(≠⊆⍺) ⍘ ⍺ → ⊣
```

Address arguments in tacit functions

```
(≠⊆⊣)
```

Address arguments in tacit functions

```
'12,3,456'(≠⊆⊣)','
```

Hooks: X f (g X) and (f X) g X

`(⊢-⌊≠)N`	Normalise so minimum is 0
`(∪⍳⊢)Y`	Consecutive IDs
`(⊢≡∪)Y` `(∪≡⊢)Y`	All distinct

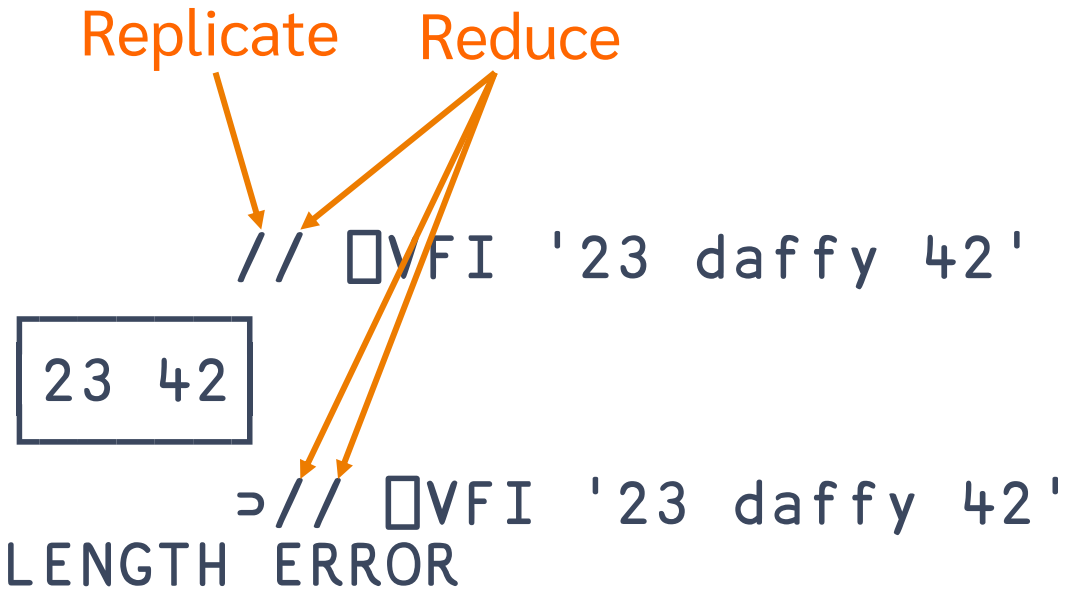
Force hybrid (/≠\⍀) to be a function

```
      ⎕VFI '23 daffy 42'
```


```
      // ⎕VFI '23 daffy 42'
```


```
      ⊃// ⎕VFI '23 daffy 42'
LENGTH ERROR
```

Force hybrid (/≠\≒) to be a function



Force hybrid (/⌿\⍀) to be a function

```
      ⊃// ⎕VFI '23 daffy 42'
LENGTH ERROR

      ⊃⊢⍤⍥⍟// ⎕VFI'23 daffy 42'
23 42
```

Remember Unique Mask?

```
      (≠'Mississippi') ≠ 'Mississippi'
Misp
      (≠≠⊢)'Mississippi'
1
      (≠⊢⍤≠⊢)'Mississippi'
Misp
```

Making values non-shy

```
      2 ⎕NQ # 'GetEnvironment' 'MAXWS'

      ⊢2 ⎕NQ # 'GetEnvironment' 'MAXWS'
4G
      vec ← ⍳10
      ⊢vec ← ⍳10
1 2 3 4 5 6 7 8 9 10
```

Making values non-shy

```
      Env←2 ⎕NQ # 'GetEnvironment',⊂
      Env 'MAXWS'

      Env←⊢2 ⎕NQ # 'GetEnvironment',⊂
      Env 'MAXWS'
4G
```

Making values non-shy

```
      Env←{2 ⎕NQ # 'GetEnvironment' ⍵}
      Env 'MAXWS'

      Env←{⊢2 ⎕NQ # 'GetEnvironment' ⍵}
      Env 'MAXWS'
4G
```

First / Last

```
      ⊢vec←⎕A
ABCDEFGHIJKLMNOPQRSTUVQXYZ

      ⊣/vec
A
      ⊢/vec
Z
```

First / Last of the First / Last Axis

```
      mat
abc
def
ghi
```

```
      ⊣≠mat
abc
      ⊢≠mat
ghi
```

```
      ⊣/mat
adg
      ⊢/mat
cfi
```

Exercise - Last column

Using `]runtime` and `mat←?100 100⍴1E6`

- Compare at least 3 ways to return the last column of `mat` as a vector
- Make sure you use `⊢/mat` as one of your choices

```
]runtime -c ⊢/mat ,¯1↑[2]mat mat[;2⊃⍴mat] (⌽mat)[;1]

⊢/mat          → 1.4E¯7 |     0% ⎕
,¯1↑[2]mat     → 8.6E¯7 |  +523% ⎕⎕⎕⎕⎕⎕⎕⎕
mat[;2⊃⍴mat]   → 3.5E¯7 |  +154% ⎕⎕⎕
(⌽mat)[;1]     → 4.2E¯6 | +2973% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
```

Propagate missing left argument of dfn

```
      Div←{0=⍬NC'⍺': ÷⍵ ⋄ ⍺÷⍵}
      Div←{⍺←⊢ ⋄ ⍺÷⍵}

      Div 3
0.3333333333

      4 Div 3
1.33333333
```

Counteracting dfn auto-localization

Modify a global variable:

```
      g←10
      ∇ foo y
        g←2×y
      ∇
      foo 3
      g
6
```

```
      g←10
      {g←2×⍵} 3
      g
10
      {g⊢←2×⍵} 3
      g
6
```

Counteracting dfn auto-localization

Use a pass-through value:

```
      F←-
      ∇ r←Use y
        r←y,F y←2×y
      ∇
      Use 3
6 ¯6
```

```
6 6
```

```
      F←-
      {y,F y←2×⍵} 3
      {y,F⊢y←2×⍵} 3
6 ¯6
```

Counteracting dfn auto-localization

Modified assignment:

```
      g←10 ⋄ F←-
      ∇ Mod y
        g F←y
      ∇
      Mod 2
      g
8
```

```
      g←10 ⋄ F←-
      {g F←⍵} 2
      g
10

      {g F∘⊢←⍵} 2
      g
8
```

Create matrix of identical rows/columns

```
      'ABC'∘.⊢'abcd'
abcd
abcd
abcd
      'ABC'∘.⊣'abcd'
AAAA
BBBB
CCCC
```

Inline "if"

```
      true_case ⊣⍤(condition)⊢ false_case

      'true'⊣⍤(3>4)⊢'false'
false
      'true'⊣⍤(9>4)⊢'false'
true
```

Left/Right/Same Exercise

Given 2 matrices, M and N, where M has the same number of rows as N has columns – write an expression to multiply the last column of M by the first row of N.

```
      1   2   3  (your-expression) 1 2 3 4
      4   5   6                    5 6 7 8
      7   8   9
     10  11  12

3 12 27 48
```

Left/Right/Same Exercise Solution

1. `(⊢/⍤¨⍤1 ×⍤1 ≠⍤¨⍤1 ⊢)`
   `{(⊢/⍺)×⍤1 ≠⍤1 ⍵}`

Where

Where: definition

```
      Y                  1 0 1 1 0 0 1 0 1
      ⍳Y                 1 3 4 7 9
      ⍳⍴Y                1 2 3 4 5 6 7 8 9
(,Y)/(,⍳⍴Y)              1   3 4     7   9
```

Where: definition

```
      Y    1 0 1 1 0 0 1 0 1
      ⍳Y   1 3 4 7 9
    ⍳⍴Y    1 2 3 4 5 6 7 8 9
(,Y)/(,⍳⍴Y)  1 3 4 7 9
```

```
0≤Y
```

Where: Rank > 1

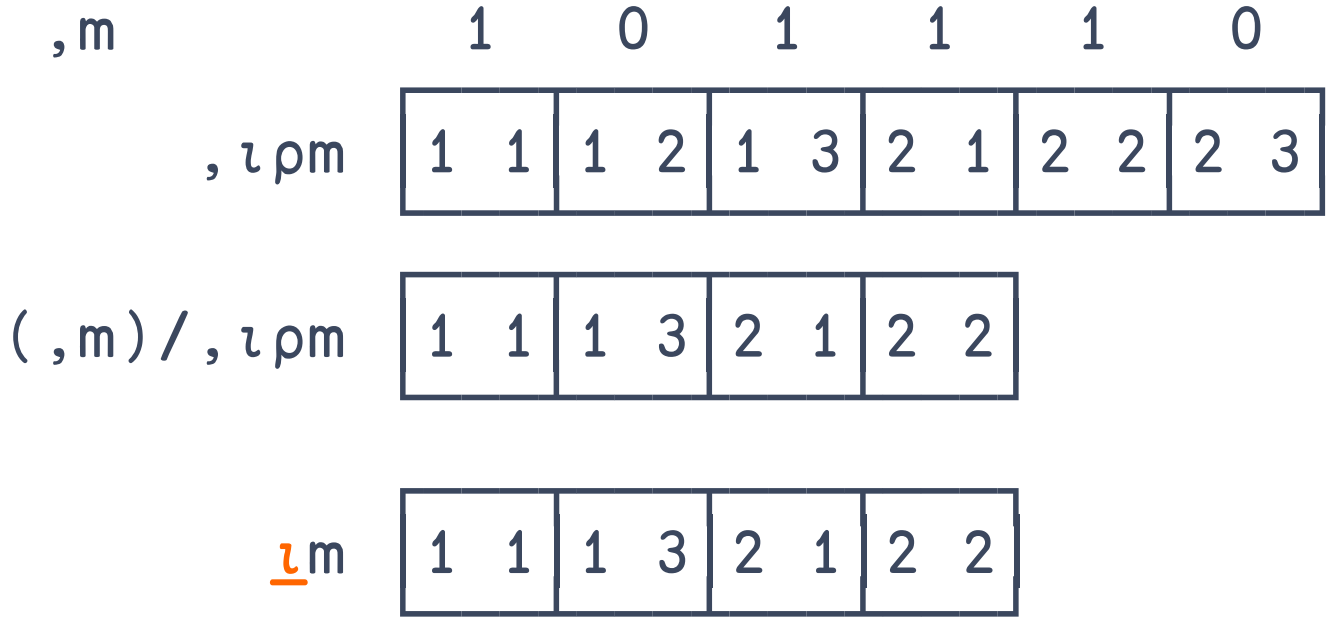
```
      m ← 2 3 ⍴ 1 0 1 1 1 0
```

```
   m
1  0  1
1  1  0
```

```
      ⍳⍴m
```

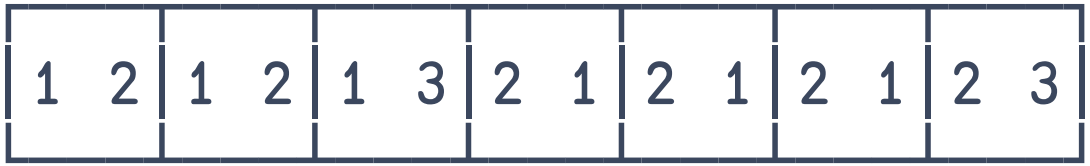
1	1	1	2	1	3
2	1	2	2	2	3

Where: Rank > 1



Where: Y > 1

```
      ⍳ 1 0 2 0 3
1 3 3 5 5 5
      ⊢ m ← 2 3 ⍴ 0 2 1 3 0 1
0 2 1
3 0 1

      ⍳m
```



Where: what about?

```
      ⍳5  ⍝ what will this produce?
```

```
      ⍳,5 ⍝ how about this?
1 1 1 1 1
```

Where: usage

```
      price←71 82 81 82 84 59
      (75≤price)/price
82 81 82 84
      (75≤price)/⍳≢price
2 3 4 5
      ⍸75≤price
2 3 4 5
      price[⍸75≤price]
82 81 82 84
```

Where: Selection

```
      fruit←'Apple' 'Banana' 'Cherry' 'Date'
      select←1 1 0 1

      select/fruit
 Apple  Banana  Date
      ⍳select
1 2 4
      fruit[⍳select]
 Apple  Banana  Date
```

Where: Multi-selection

```
      fruit←'Apple' 'Banana' 'Cherry' 'Date'
      select←1 2 0 1

      select/fruit
Apple  Banana  Banana  Date

      fruit[⍸select]
Apple  Banana  Banana  Date
```

Where: Multi-dimensional selection

```
      spice ← 'Anise' 'Basil' 'Chili' 'Dill'
      ⊢stuff← ↑fruit spice
 Apple  Banana  Cherry  Date
 Anise  Basil   Chili   Dill
      ⊢select←↑(1 2 0 1) (0 0 0 2)
1 2 0 1 0
0 0 0 2 0
      select/stuff
RANK ERROR
```

Where: Multi-dimensional selection

```
      spice ← 'Anise' 'Basil' 'Chili' 'Dill'
      ⊢stuff← ↑fruit spice
 Apple  Banana  Cherry  Date
 Anise  Basil   Chili   Dill
      ⊢select←↑(1 2 0 1) (0 0 0 2)
1 2 0 1 0
0 0 0 2 0
      stuff[⍸select]
 Apple  Banana  Banana  Date  Dill  Dill
```

Where Inverse: ⍸⍣¯1

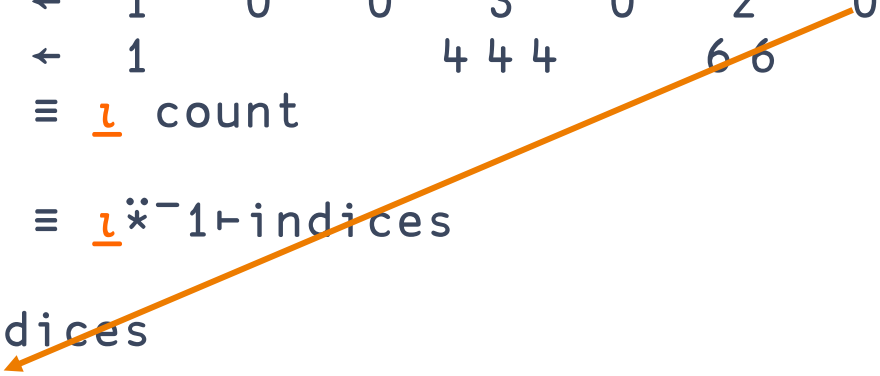
```
      ⍸ Y          ⍝ counts  → indices
      ⍸ 2 0 2
1 1 3 3

      ⍸⍣¯1⊢ Y      ⍝ indices → counts
      ⍸⍣¯1⊢ 1 1 3 3
2 0 2
```

Where: Representing a multiset

```
      all     ← 'a' 'b' 'c' 'd' 'e' 'f'
      count   ←  1   0   0   3   0   2
      indices ←  1           4 4 4     6 6
      indices ≡ ⍳ count
1
      count   ≡ ⍳⍤¨¯1⊢indices
1
```

Where Inverse:

```
      all     ← 'a' 'b' 'c' 'd' 'e' 'f' 'g'
      count   ←  1   0   0   3   0   2   0
      indices ←  1           4 4 4       6 6
      indices ≡ ⍳ count
1
      count   ≡ ⍳⌸¯1⊢indices
0
      ⍳⌸¯1⊢indices
1 0 0 3 0 2 ?
      count   ≡ (⍴count) ↑ ⍳⌸¯1⊢indices
1
```



Where Inverse: Reconstruction of masks

```
      data ← 'banana'
      (∪∘.≡⊢) data
1 0 0 0 0 0
0 1 0 1 0 1
0 0 1 0 1 0
```

Where Inverse: Reconstruction of masks

```
      data ← 'banana'
      ('≡',∪data) , data ⍪ (∪∘.≡⊢)data
≡ b a n a n a
b 1 0 0 0 0 0
a 0 1 0 1 0 1
n 0 0 1 0 1 0
```

Where Inverse: Reconstruction of masks

```
      ⊢⍸ d←'banana'
1 0 0
2 4 6
3 5 0
      ⍸⍣¯1⍥⊢⍸ d
1 0 0 0 0 0
0 1 0 1 0 1
0 0 1 0 1 0
```

```
      {⍸⍣¯1⊢⍞←⍵}⍥⊢⍸ d
1
2 4 6
3 5
1 0 0 0 0 0
0 1 0 1 0 1
0 0 1 0 1 0
```

Where Inverse: Reconstruction of masks

```
      data ← 10000⍴'Mississippi'
      ]runtime -c (∪⍤.≡⊢)data  ⍸⍤¯1⍤⊢⌸data

(∪⍤.≡⊢)data → 2.5E¯3 |   0% ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
⍸⍤¯1⍤⊢⌸data → 2.4E¯4 | -91% ▯▯
```

Where: Exercise

Given a Boolean vector (`v←¯1+?100⍴2`), write an expression using `⍳` (twice!) to return the highest number of consecutive 1's.

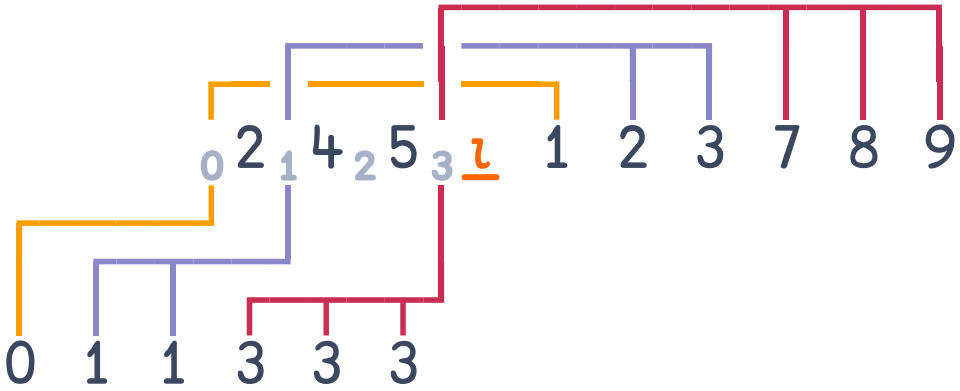
```
      (yours) 1 0 1 1 1 0 1 1 1 1 0 1 1 0 1
4

      ⌈/¯2-/⍳1≠0,⍨¯2-/⍳
```

Where: Exercise Solution

```
      V←1 0 1 1 1 0 1 1 1 1 0 1 1 0 1
      ⌈/¯2-/⍳1≠0,⍨¯2-/⍳V
4
```

Interval Index

Interval Index



Interval Index – the bucket list primitive

```
          Breaks          ⍳  Data
           10 20 30       ⍳  23 2 42 7 5 19 24
¯∞ ←---10---20---30--→ ∞  Breaks define buckets
    0     1     2     3      Buckets/Intervals
                             (⎕IO←1)
```

0	1	2	3
[¯∞,10)	[10,20)	[20,30)	[30,∞]
D<10	D<.<10 20	D<.<20 30	D≥30

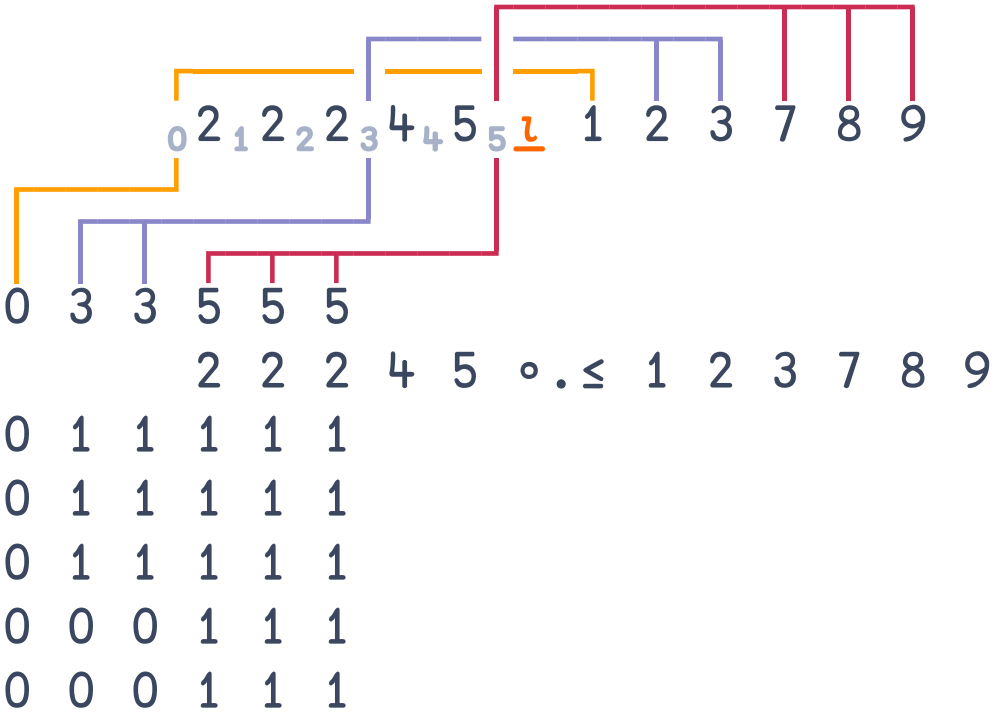
What if you want (lower,upper]?

```
      2 4 5 ⍳ 1 2 3 5 7 8 9
0 1 1 3 3 3 3
      2 4 5 ∈⍨ 1 2 3 5 7 8 9
0 1 0 1 0 0 0
      2 4 5 (⍳-∈⍨) 1 2 3 5 7 8 9
0 0 1 2 3 3 3
```

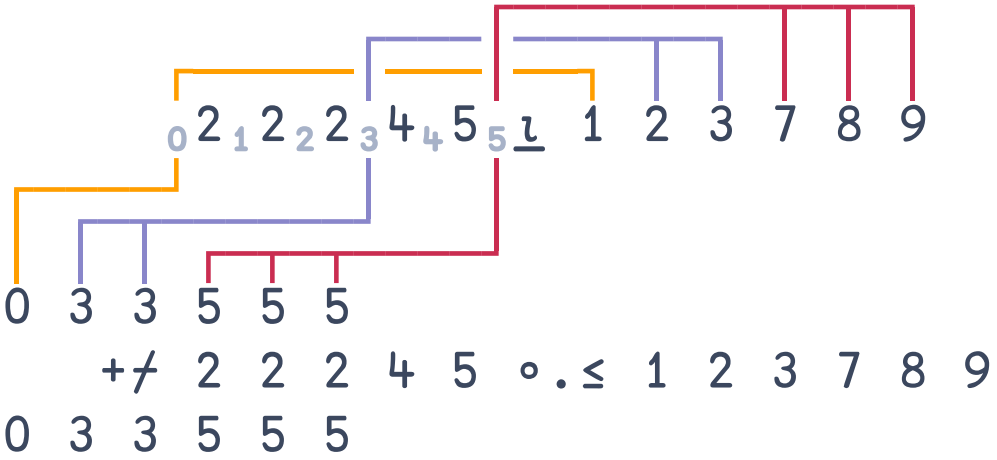
Check if value is in interval

```
      2 5 ∘.≤ 1 2 3 4 5 6
0 1 1 1 1 1
0 0 0 0 1 1
      ≥/2 5 ∘.≤ 1 2 3 4 5 6
0 1 1 1 0 0
      1=2 5 ⍳ 1 2 3 4 5 6
0 1 1 1 0 0
      2 5 (1=⍳) 1 2 3 4 5 6
0 1 1 1 0 0
```

Non-unique cut-offs



Non-unique cut-offs



Performance

```
      n←?⍨10000
      c←{⍵[⍋⍵]}100?10000

    ]runtime -c +⌿c∘.≤n c⍸n

+⌿c∘.≤n → 9.9E¯5 |    0% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
c⍸n     → 9.2E¯6 | -91% ⎕⎕⎕⎕
```

Buckets of higher rank

```
      qtrs
2024   1 15
2024   4 15
2024   7 15
2024 10 15
```

```
      dts
2024   3  6
2024   9  8
2024   7 28
2024 10 21
2024   8 15
2024   7 11
```

```
      qtrs⍸dts
1 3 3 4 3 2
```

Total Array Ordering

```
      ⊢qtrs←⍋qtrs
```

2024 1 15	2024 4 15	2024 7 15	2024 10 15

```
      ⊢dts←⍋dts
```

2024 3 6	2024 9 8	2024 7 28	2024 10 21	202 ...

```
      qtrs ⍳ dts
1 3 3 4 3 2 4 4 1 1
```

Total Array Ordering

```
      'Abe' 'Bob' 'Carla' 'Dana'⍳'Bea' 'Aaron'
1 0
```

Total Array Ordering

```
      ⊢table←'Abe' 'Abe' 'Abe' 'Bob',⍪1 3 9 2
```

Abe	1
Abe	3
Abe	9
Bob	2

Total Array Ordering

```
      new← 'Abe' 4
      pos← table⍳new
2
      pos↑table
```

Abe	1
Abe	3

```
      pos↓table
```

Abe	9
Bob	2

table

Abe	1
Abe	3
Abe	9
Bob	2

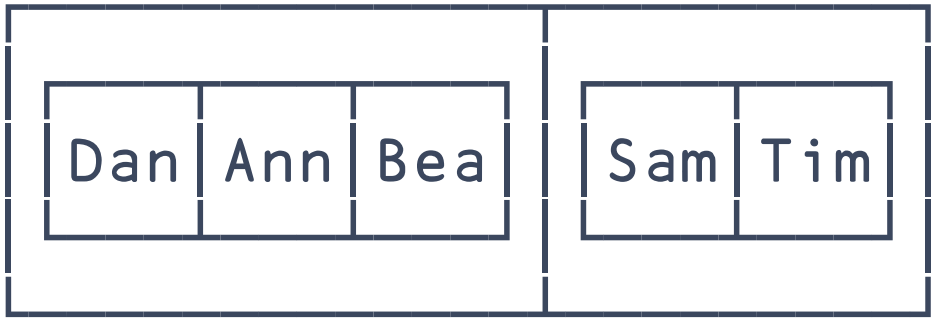
Total Array Ordering

```
      pos(↑;new;↓)table
```

Abe	1
Abe	3
Abe	4
Abe	9
Bob	2

Group by interval

```
        'AL'⍳'Dan' 'Sam' 'Ann' 'Tim' 'Bea'
1 2 1 2 1
        n←'Dan' 'Sam' 'Ann' 'Tim' 'Bea'
        ('AL'⍳n){⊂⍵}⌸n
```



Interval Index Exercise

Many schools convert numeric scores into a letter grade. Scotland uses:

Percent	Grade	Description
70–100%	A	Highest grade
60–69%	B	Very strong pass
50–59%	C	Pass and be accepted by universities
40–49%	D	Borderline: the student should re-sit the course
0–39%	No Award	The student has failed the course

Write a function using `⍸` that will return the grade for a given percentage.

Interval Index Exercise Solution

This solution also deals with scores that are less than 0 or greater than 100.

```
 GradeFor←{
     grades←,⍕'Invalid' 'No Award' 'D' 'C' 'B' 'A' 'Invalid'
     grades⊃⍨1+(100=⍵)-⍨0 40 50 60 70 100⍸⍵
 }

       GradeFor ¯1 0 39 40 49 50 59 60 69 70 99 100 101
Invalid  No Award  No Award  D  D  C  C  B  B  A  A  A  Invalid
```

Key

Key

Monadic Operator

Analogous to SQL "GROUP BY"

SELECT (dates,amounts) FROM sales GROUP BY customer

Monadic Key: f⌸Y

```
{α ω}⌸'Abracadabra'
```

A	1
b	2 9
r	3 10
a	4 6 8 11
c	5
d	7

ω: indices where α occurs

α: ∪Y

Dyadic Key: X f⌸Y

```
      'Abracadabra'{α ω}⌸'ABCDEFGHIJK'
```

A	A
b	BI
r	CJ
a	DFHK
c	E
d	G

ω: elements of Y where α occurs

α: ∪X

Dyadic Key

```
'Abracadabra'{⍺ ⍵}⌸⍳≠'Abracadabra'
```

A	1
b	2 9
r	3 10
a	4 6 8 11
c	5
d	7

```
{⍺ ⍵}⌸  'Abracadabra'
```

Key: grouping by an aspect of the data

```
(⊂C'Abracadabra'){α ω}⌸'Abracadabra'
```

a	Aaaaa
b	bb
r	rr
c	c
d	d

Key: grouping by an aspect of the data

```
      ⊢v←?10⍴20
2 11 19 14 1 14 3 8 1 16

      (3|v){⍺ ⍵}⌸v
```

2	2 11 14 14 8
1	19 1 1 16
0	3

f⍇ - What the f?

```
      v←?1000⍴10
      {⍺ ('□'⍴⍨⌊.5+.1×≠⍵)}⍇v
 4  □□□□□□□□□□
10  □□□□□□□□□□
 2  □□□□□□□□□□
 8  □□□□□□□□□□
 1  □□□□□□□□□□□
 5  □□□□□□□□□□□
 9  □□□□□□□□□
 3  □□□□□□□□□
 6  □□□□□□□□□
 7  □□□□□□□□□□
```

Key versus Partition

```
      t←'Herebedragons'
      p←1  1  1  1  2  2  3  3  3  3  3  3  3

      p{⊂⍵}⌸t
```

Here	be	dragons

```
      p⊆t
```

Here	be	dragons

Key versus Partition

```
t←'Herebedragons'
p←3 3 3 3 2 2 3 3 3 3 3 3 3

p{⊂ω}⌸t
```

Heredragons	be

```
p⊆t
```

Herebe	dragons

Key versus Unique

```
      {αω}⌸'abracadabra'
```

a	1 4 6 8 11
b	2 9
r	3 10
c	5
d	7

```
      {α}⌸'abracadabra'
abrcd

      ∪'abracadabra'
abrcd
```

Key issue: missing keys

```
      d←'2718281828'

      {α,≠ω}⌸d
2  3
7  1
1  2
8  4
```

```
      {α,≠ω}⌸⍬D,d
0  1
1  3
2  4
3  1
4  1
5  1
6  1
7  2
8  5
9  1
```

Key issue: missing keys

```
      d←'2718281828'

      {⍺,≠⍵}⌸d
2 3
7 1
1 2
8 4
```

```
      {⍺,¯1+≠⍵}⌸⎕D,d
0 0
1 2
2 3
3 0
4 0
5 0
6 0
7 1
8 4
9 0
```

Key issue: sorting keys

```
v←?10⍴20
(3|v)⌸{⍺ ⍵}⍸⍳≠v
```

1	1 6 7 8 10
2	2 4 9
0	3 5

```
sort←⊂⍤⍋⌷⊢
sort(3|v)⌸{⍺ ⍵}⍸⍳≠v
```

0	3 5
1	1 6 7 8 10
2	2 4 9

Key issue: sorting keys

```
(3|v){α ω}⌸⍳≠v
```

1	1 6 7 8 10
2	2 4 9
0	3 5

Key issue: sorting keys

```
(0 1 2,3⍴v){⍺ ⍵}⌸0 1 2,⍳≠v
```

0	0 3 5
1	1 1 6 7 8 10
2	2 2 4 9

Key issue: sorting keys

```
(0 1 2,3⍴⍵){⍺ ⍵}⌸0 1 2,⍳≢⍵
```

0	0 3 5
1	1 1 6 7 8 10
2	2 2 4 9

Key issue: sorting keys

```
(0 1 2,3|v){α (1↓ω)}⌸0 1 2,⍳≢v
```

0	3 5
1	1 6 7 8 10
2	2 4 9

Key Exercise

You may have heard there's an election coming up in the USA. Most polling locations have designated tables for ranges of the alphabet.

For instance, if surname/last name happens to be "Becker", you need to go to table 1 to have your registration validated.

Key Exercise

Write a function to group a list of non-case sensitive surnames by table as follows:

Surname Begins With	Go To Table
A–H & a–h	1
I–O & i–o	2
P–Z & p–z	3
Anything else	4

Key Exercise Solution

This solution returns a matrix sorted by table number with each row containing the list of names assigned to each table.

```
      tables←{((⍳4),4@(=∘0)⊢{('z'=⍵)-⍨'aipz'⍳⍵}⌸⊂⍨⍵){⍺ (1↓⍵)}⌸1 2 3 4,⊆⍵}
      tables '123' 'becker' 'smith' 'zither' 'ørson'
1   becker
2
3   smith  zither
4   123  ørson
```

This solution returns a matrix not sorted by table number and only rows where a name exists for a particular table.

```
      tables2←{(4@(=∘0)⊢,{('z'=⍵)-⍨'aipz'⍳⍵}⌸⊂⍨⍵){⍺⍵}⌸,⍵}
      tables2'123' 'becker' 'smith' 'zither' 'ørson'

4   123  ørson
1   becker
3   smith  zither
```

Key Exercise Solution

```
tables←{((⍳4),4@(=∘0)⊢{('z'=⍵)-⍨'aipz'⍸⍵}⌸C⊃¨⍵){⍺ (1↓⍵)}⌸1 2 3 4,⊆⍵}

      tables '123' 'becker' 'smith' 'zither' 'ørson'
1   becker
2
3   smith  zither
4   123  ørson

      tables2←{(4@(=∘0)⊢,{('z'=⍵)-⍨'aipz'⍸⍵}⌸C⊃¨⍵){⍺⍵}⌸,⍵}
      tables2'123' 'becker' 'smith' 'zither' 'ørson'

4   123  ørson
1   becker
3   smith  zither
```

At

At

```
{X}(what @ where)Y
2(*⍨ @ 2 4 6)⍳6
1 4 3 16 5 36
```

At: Separating the argument

```
      '*' @ 1 3 5 'Hello'
```


```
      '*' @ 1 3 5 ⊢ 'Hello'
*e*l*
      ('*' @ 1 3 5) 'Hello'
*e*l*
```

At: versus Selective Assignment

```
      str←'Hello'
      mask←str∊'lo'
      (mask/str)←'*'
      'abc ',str
abc He***

      'abc ','*'@(∊∘'lo')⊢'Hello'
abc He***
```

At within dfns

```
      'abc ', 'lo'{(⍵∊⍺)/⍵)←'*'}'Hello'
SYNTAX ERROR

      'abc ', 'lo'{w←⍵ ⋄ ((w∊⍺)/w)←'*'}'Hello'
abc **
```

At within dfns

```
      'abc ', 'lo'{w←ω◇((w∊α)/w)←'*'◇w}'Hello'
abc He***

      'abc ','lo' {'*'@(∊∘α)⊢ω} 'Hello'
abc He***
```

At versus Indexed Assignment

```
      23,(⍳10)[3 5 7] ← 42
SYNTAX ERROR
      vec←⍳10
      23,vec[3 5 7]←42
23 42
      23,vec
23 1 2 42 4 42 6 42 8 9 10
      23,42@3 5 7 ⊢ ⍳10
23 1 2 42 4 42 6 42 8 9 10
```

At versus Modified Assignment

```
      vec←5⍴0
      ⊢inds←?20⍴5
5 3 3 3 2 4 1 4 1 2 2 3 2 5 3 5 1 1 4 1
      vec[inds]+←1
      vec
5 4 5 3 3
      1+@inds⊢5⍴0
1 1 1 1 1
```

The whats and wheres of @

```
(array @ array)

(array @ function)

(function @ array)

(function @ function)
```

Major Cell Orientation

```
      r1←10×⍳3
      42 @ 2 ⊢ r1
10 42 30

      r2←2 3⍴10×⍳6
      42 @ 2 ⊢ r2
10 20 30
42 42 42
```

```
      r3←2 3 3⍴10×⍳18
      42 @ 2 ⊢ r3
10 20 30
40 50 60

42 42 42
42 42 42
```

When the left operand (`f`) is a function...

`f` is applied to the sub-array defined by `g`

```
      +/(⍳10)[2 3 4 6 9]
24
      +/@2 3 4 6 9 ⊢ ⍳10
1 24 24 24 5 24 7 8 24 10
```

When the left operand (`f`) is a function...

```
      +\(⍳10)[2 3 4 6 9]
2 5 9 15 24
```

When the left operand (`f`) is a function...

```
      +\(⍳10)[2 3 4 6 9]
  2 5 9   15     24
      +\@2 3 4 6 9 ⊢ ⍳10
1 2 5 9 5 15 7 8 24 10
```

{X}(f @ N) Y

N is a scalar or vector of indices into Y

f is a function to be applied on the sub-array defined by N

```
      42 (×@ 3 5 7) ⍳10
1 2 126 4 210 6 294 8 9 10
      42×(@3 5 7) ⍳10
      42×@3 5 7 ⊢ ⍳10
```

{X}(f @ N) Y

```
      ⊢m←2 4⍴4/2 3
2 2 2 2
3 3 3 3
      m×@2 4 ⊢ d
```

```
      ⊢d←4 4⍴⍳16
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

{X}(f @ N) Y

```
      ⊢m←2 4⍴4/2 3
2 2 2 2
3 3 3 3
      m×@2 4 ⊢ d
 1  2  3  4
10 12 14 16
 9 10 11 12
39 42 45 48
```

```
      ⊢d←4 4⍴⍳16
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

{X}(f @ N) Y

```
      ϕ@1 3⊢'apple' 'berry' 'cherry'
 cherry  berry  apple

      ϕ¨@1 3⊢'apple' 'berry' 'cherry'
 elppa  berry  yrrehc

      2ϕ@1 3 4⊢'apple' 'banana' 'berry' 'cherry'
cherry  banana  apple  berry
```

{X}(f @ g) Y

g is a function that returns a Boolean array of shape ⍴Y

f is function to apply to elements of Y at indices ⍸g Y

```
      (⌽⍤@(2|≢⍤))'dyalog' 'apl' 'is' 'great'
 dyalog  lpa  is  taerg

      (⌽@(2|≢⍤))'dyalog' 'apl' 'is' 'great'
 dyalog  great  is  apl
```

At Exercise

Given an integer vector, double the first occurrence of the smallest number.

```
      (your solution)3 1 4 1 5 9
3 2 4 1 5 9
```

At Exercise Solution

```
      2×@(<\⊢=⌊/)⊢3 1 4 1 5 9
3 2 4 1 5 9
```

Constant

Constant

```
      constant←⍤⍤
      ⎕NC 'constant'
4 ⍝ operator

      seven←7⍤⍤
      ⎕NC 'seven'
3 ⍝ function
```

Constant

```
      'Hi'(1 2 3⍨)'Earth'
1 2 3

      (1 2 3⍨)'Hello'
1 2 3
```

Constant with other operators

```
      42⍨⍨'Life' 'Universe' 'Everything'
42 42 42
      (?9)⍨⍨1 2∘.×1 2 3 4
7 7 7 7
7 7 7 7
```

Constant in trains

normal code	`(⍺+⍵)*3`
invalid train	`+*3`
workaround	`3*⍨+`
workaround	`*∘3+`

Constant in trains

normal code	`(α+ω)*3`
invalid train	`+*3`
valid train	`+*3¨~`

Constant in trains

normal code	`(α+ω)*÷3`
invalid train	`+*∘÷3`
workaround	`(÷3)*⍨+`
workaround	`3*∘÷⍨+`

Constant in trains

normal code	`(⍺+⍵)*÷3`
invalid train	`+*∘÷3`
valid train	`+*∘÷3¨`
valid train	`+*1÷3¨`

Constant in operands

```
      mask←1 0 0 0 1 0 0 ⋄ data←'AbcdEfg'

      '□'@{mask}data
□bcd□fg

      mask{'□'@{α}ω}data
VALUE ERROR
      mask{'□'@{α}ω}data
                 ^
```

Constant in operands

```
      mask←1 0 0 0 1 0 0 ⋄ data←'AbcdEfg'

      '□'@{mask}data
□bcd□fg

      mask{'□'@(α∵)ω}data
□bcd□fg
```

Constant Exercise

Given a matrix M, return 42 for every column.

```
      (your_solution) 3 4⍴⎕A
42 42 42 42
```

Constant Exercise Solution

```
      4 2 ⍨ ≠ 3  4 ⍴ ⍬ A
42 42 42 42
```

f⌸ ≠Y

X⍸Y A⍨

⍸Y f@g f@B A@g A@B

≢Y X⊣Y X⊢Y ⊢Y ⊣Y