DYALOG

Code Golf Hackathon

Elsinore 2017

Jay Foad
Adám Brudzewsky
Marshall Lochbaum
Roger Hui

FizzBuzz

Given an array of integers,

- replace all the one that are divisible by 3 with `'Fizz'`
- all the ones that are divisible by 5 with `'Buzz'`
- all the ones that are divisible by both with `'FizzBuzz'`

```
      FizzBuzz←{∨/d←0=4/3 5|⍵:d/'FizzBuzz'◊⍵}¨

      FizzBuzz 7+⍳8
8  Fizz  Buzz  11  Fizz  13 14  FizzBuzz  16
```

Function Trains

A sequence of 2 or more functions *in isolation* is a train:

```
      + , - , × , ÷   ⍝ 7 functions
```

The functions can be primitive, derived or defined:

```
      foo +.× goo     ⍝ 3 functions
```

Function Trains: *in isolation*

```
      3 +,- 2      ⍝ not a train
1
      3(+,-)2      ⍝     a train
5 1
      (+,-)/3 2    ⍝ train reduction
 5 1
```

Function Trains: *in isolation*

```
      f ← +,-   ⍝ a train
      3 f 2     ⍝ train application
5 1
```

Function Trains: definition

The functions in a train are grouped in threes, starting from the right:

```
+,-,×,÷   ↔   +,(-,(×,÷))
```

There may be a group of two left over:

```
-  +⌿  ÷  ≢   ↔   -(+⌿ ÷ ≢)
```

A useful diagnostic tool: `]box on -trains=parens`

Function Trains: definition

A 2-train is an *atop*:

```
      (g h) ω  ←→      g (  h ω)
    α (g h) ω  ←→      g (α h ω)
```

A 3-train is a *fork*:

```
      (f g h) ω  ←→  (  f ω) g (  h ω)
    α (f g h) ω  ←→  (α f ω) g (α h ω)
```

The left tine of a fork may also be an array:

```
      (A g h) ω  ←→    A g (  h ω)
    α (A g h) ω  ←→    A g (α h ω)
```

Function Trains: atops

```
      2 (?⍴) 6
←→    ? (2 ⍴ 6)
←→    ? 6 6
←→    5 1   ⍝ for example!
```

Function Trains: forks

```
      (+⌿ ÷ ≢) 1 2 3 4
↔     (+⌿ 1 2 3 4) ÷ (≢ 1 2 3 4)
↔     10 ÷ 4
↔     2.5
```

Function Trains: long trains

Alternate functions starting from the last are applied to the train's argument(s):

```
      6 (+   ,   -   ,   ×   ,   ÷) 2
        (6+2),(6-2),(6×2),(6÷2)
          8   ,   4   ,   12  ,   3
```

Intervening functions are applied between these results

CosmicRay

Given a Boolean array, flip a random bit.

```
      CosmicRay←{~@(⊂?⍴⍵)⊢⍵} ⍝ from 16.0
      CosmicRay 4 4⍴2|⎕AVU
0 0 0 1
0 0 0 1
1 1 1 1
1 1 0 1
```

Marinus' shorter solution which takes 200–∞ as long:

```
      CosmicRay←⊢≠⍳∘⍴∊⊂∘?∘⍴
```

At operator

```
(mod @ sel) ⍵  ⍝ say: "mod at sel in ⍵"
```

Right operand `sel` selects items of ⍵ to modify (or replace).

Left operand `mod` says how to modify them (or what to replace them with).

```
      100 ×@(2∘|) ⍳7          ⍝ "100-fold at odd items"
100 2 300 4 500 6 700
      ('*' @ 2 4) 'Hello'     ⍝ "* at 2 4" (replacement)
H*l*o
```

At operator: operands

Left operand (modifier) can be either:

- An *array* of replacement values
- A *function* used to modify values

Right operand (selector) can be either:

- A *simple array* selecting major cells
- A *nested array* for choose or reach indexing
- A *function*, applied to *⍵*, returning a Boolean mask used for scatterpoint indexing

At operator: left operand

Left operand (modifier) can be either:

- An *array* of replacement values
- A *function* used to modify values

```
      (200 400@2 4) ⍳5   ⍝ array
1 200 3 400 5
      (⌽@2 4) ⍳5         ⍝ monadic function
1 4 3 2 5
      100 (×@2 4) ⍳5     ⍝ dyadic function
1 200 3 400 5
```

At operator: right operand

Right operand (selector) can be either:

- A *simple array* selecting major cells
- A *nested array* for choose or reach indexing
- A *function*, applied to ⍵, returning a Boolean mask used for scatterpoint indexing

```
      100@1 3 5 ⍳9          ⍝ simple array
100 2 100 4 100 6 7 8 9
      100@(2∘|) 3 3⍴⍳9      ⍝ Boolean function
100   2 100
  4 100   6
100   8 100
```

At operator: syntax

Dyadic operators with array right operand often present a parsing problem:

```
      ⊂⍤9 'Hello'  ⍝ Wrong! Parses as ⊂⍤(9 'Hello')
```

Use parentheses:

```
      (⊂⍤9)'Hello'
```

Or save one character by inserting an identity function:

```
      ⊂⍤9⊢'Hello'
```

But none of this is needed if ⊂⍤9 is followed by another function or end of line:

```
      ⊂⍤9 ⌽'Hello'
      f←⊂⍤9
```

UniqueSums

Given a vector, group identical items together, then sum each group.

```
      UniqueSums←×∘≠⊒ ⍵ from 14.0
      UniqueSums 3 1 4 1 5 9 2 6 5 4
3 2 8 10 9 2 6
      UniqueSums 2 7 1 8 2 8 1 8 2 8
6 7 2 32
```

Key operator

α f⌸ ω

Left argument α specifies keys

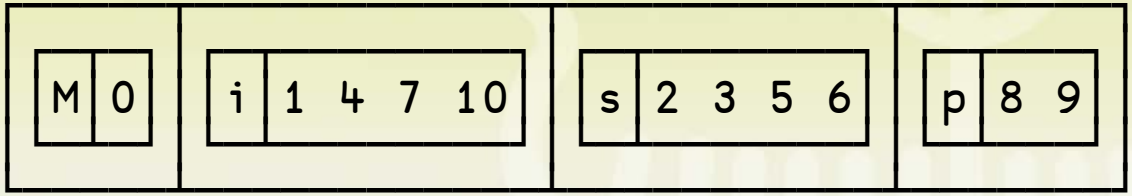
Right argument ω specifies values

For each unique key in α, f is applied to:

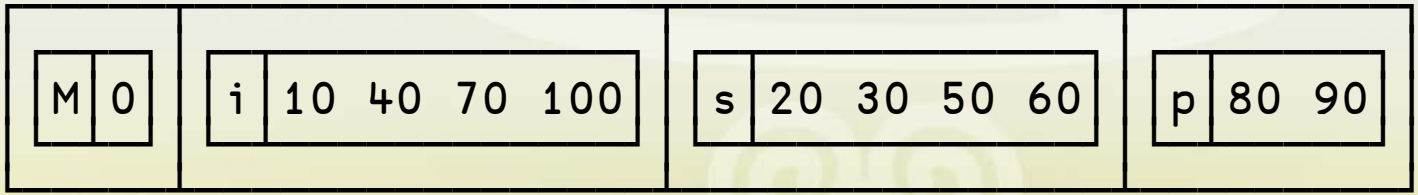
- that key (left argument)
- the values corresponding to all occurrences of that key (right argument)

Key operator: monadic and dyadic

```
      {⊂αω}⌸ 'Mississippi'
```




```
'Mississippi' {⊂αω}⌸ 10×⍳11
```



InterleaveReverse

Given a vector, interleave the even-position items with the reversed odd-position items.

```
      InterleaveReverse←⌽@{2|⍳≠⍵} ⍝ from 16.0
      InterleaveReverse 'aBcDeFgHi'
iBgDeFcHa
      InterleaveReverse 'abcdefghij'
ibgdefchaj
      InterleaveReverse ⍳10
9 2 7 4 5 6 3 8 1 10
      InterleaveReverse 'AA' 'bb' 'CC' 'dd' 'EE' 'ff' 'GG'
 GG  bb  EE  dd  CC  ff  AA
```

The Stack Exchange Code Golf Community

- codegolf.stackexchange.com
- Beware of *byte* count (⍠, ⍤, ⍇, ⍁, ⊆, ⍳ → ⎕OPT, ⎕U2364, etc.)
- You may use ⍞ and ⎕ for input
- Use tio.run to automatically format a submission
- Explain your code if you can
- The submission we made during the workshop:
  codegolf.stackexchange.com/a/142707/43319