# Train Spotting in Dyalog APL

## Richard Park (23rd January 2020)

# Trains (choo choo)

# Function trains: *a bit of history*

| Expression | | Composition | |
|---|---|---|---|
| **TMN** | **APL** | **TMN** | **APL** |
| $f(g(x))$ | f g x | $(f \circ g)(x)$ | (f∘g) x |
| $f(x) \divideontimes g(x)$ | (f x) ⋇ g x | $(f \divideontimes g)(x)$ | (f⋇g) x |

# Function Trains: *valence*

```
      (+,-)2          ⍝ monadic train
2 ¯2
     3(+,-)2          ⍝ dyadic train
5 1
```

| TMN | | APL | | TMN |
|---|---|---|---|---|
| $3 \pm 2$ | $\Rightarrow$ | (f g h) | $\Leftarrow$ | $(f \times g)(x)$ |

# Summary

## Short, pure and simple

2 train: atop – (f α g ω)

3 train: fork – ((α f ω) g (α h ω))

# Links

https://aplwiki.com/wiki/Trains

Search: aplwiki trains

https://dfns.dyalog.com/n_tacit.htm

Search: dfns tacit

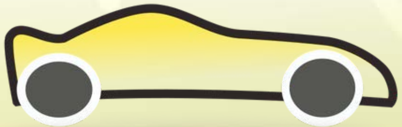
Next Webinar:

20th February 2020

(4PM UTC)

"Greasing the wheels:

Tuning APL Code for Speed" *- working title*



# Be careful

- Monadic vs dyadic

# Function Trains: *in isolation*

```
      3 +,- 2        ⍝ not a train
1
      3(+,-)2        ⍝ yes a train
5 1
      f ← +,-        ⍝ train assignment
      3 f 2          ⍝ train application
5 1
```

# Function Trains: *summary*

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* (but not an atop!) can be an array:

```
  (A g h) ω  ⇔      A g ( h ω)
α (A g h) ω  ⇔      A g (α h ω)
```

# Function Trains: *How to read*

Functions in a train are grouped in threes from right:

```
|+,-,×,÷   ⇔   (|(+,(-,(×,÷))))
```

```
      ]Box on -trains=parens    ⍝ for diagnostics
      +,-,×,÷
+,(-,(×,÷))
```

# Function Trains

Odd-numbered functions starting from the right are applied to the train's argument(s):

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

*Intervening*, even-numbered, functions are applied between results of the odd-numbered functions

# How to write

- Translating one-liners (expression / dfn)
- Direct transcription of thought concepts to code units

# Example: expression → train

- (',' ≠list)⊆list←'my,list'

# Example: dfn → train

- `',;'((~∊)⍨⊆⊢)'commas,and;semicolons'`

# Example: Thought → train

- (⌽≡⊢)
- (∪≡⊢)
- (⍉≡⊢)

# Example: Thought → train

- (+⌿÷≢)
- (+,-)
- (⊢÷+/)
- * Compound words

# When to use

- Short pure, simple functions
- Keeping related functionality together

# Short pure, simple functions

- Examples given so far…

# Keeping related functionality together

- Visually “merge” functions into a single semantic unit
- ∨/¨'some' 'thing’⍷¨⊂text
- 1 0
- 'some' 'thing'(∨/⍷)¨⊂text
- 1 0
- Alter code in fewer places
- Potential performance benefits

# Code golf

# When NOT to use

- Using compression / replicate
- 5(<{α/ω}⊢)⍳10

# Kaomoji

- https://www.jsoftware.com/papers/amuse-bouches.htm#0
- `(x>0)-(x<0)`
- `0(>-<)x`
- `1 2(∘.○)x ⍝ Not a train`
- `(⊢∘.○⊣)`
- `(≠⊆⊢)`