DYALOG

Glasgow 2024

Tacit Programming in Dyalog

Adám Brudzewsky

Rich Park

Stefan Kruger

DYALOG

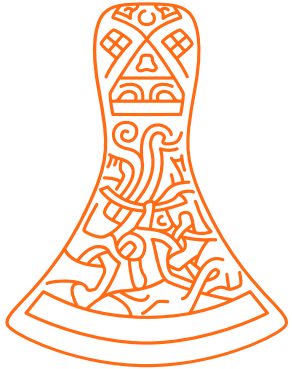
Glasgow 2024

Tacit Programming in Dyalog

Adám Brudzewsky

Rich Park

Stefan Kruger



DYALOG

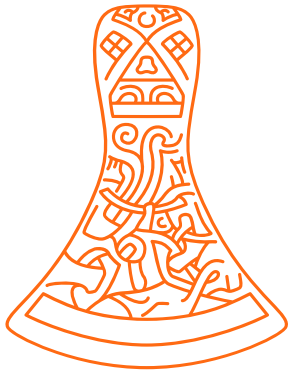
Glasgow 2024

Tacit Programming in Dyalog

Adám Brudzewsky · xpqz.github.io/cultivations/Trains

Rich Park · youtu.be/Enlh5qwwDuY “Train Spotting”

Stefan Kruger · xpqz.github.io/learnapl/tacit



What is tacit programming?

Explicit code mentions arguments:

- Expression `(⌈/N)-⌊/N←3 1 4 1 5`
- Tradfn `∇ R←Range Y …`
- Dfn `{(⌈/⍵)-(⌊/⍵)}`

Tacit code implies arguments:

- Tacit `(⌈/-⌊/)`

You already know some

```
f/    f⍨    ∘.g    f⌿    A∘g    f⌸    f⌸B
```

Operators derive tacit functions

Function composition

```
f⍤g    f⍥g    f⍨    f g h    f∘g
```

Function composition is plumbing that guides arguments to functions

Benefits

- Arguments in operands
- Memorable (like ≠⊆⊢ and +⌿÷≠)
- Adjacency (like ×- and ∨/⍷)
- Brevity (like F⍤⍥C)
- DRY (Don’t Repeat Yourself; like ≡⍤⍴)
- Just a general feeling of superiority and awesomeness

Overview

- Compositional operators (combinators)
- Trains (forks and atops)
- Tools
- Issues
- Reading

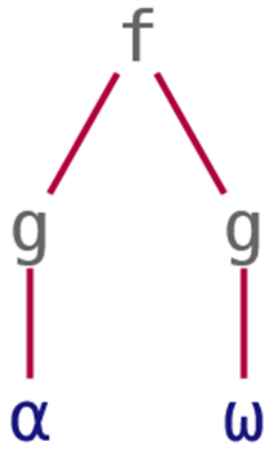
Plenty of exercises throughout

ö Over

The shape of an outer product `α ∘.f ω` is
`(⍴α) , (⍴ω)`

We can write this as
`α ,ö⍴ ω`

“pre-process both”



∘ Beside

Location of α^{th} 1 in each element of ω is

```
α ⊃¨ ⍳¨ ω
```

We can write this as

```
α ⊃∘⍳¨ ω
```

"pre-process right"


```
α f∘g ω
```

⍤ Atop

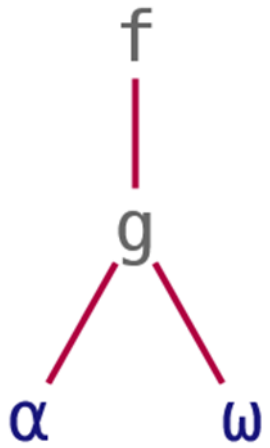
Any-presence of ⍺ in ⍵ is

```
∨/ ⍺ ⍷ ⍵
```

We can write this as

```
⍺ ∨/⍤⍷ ⍵
```

“post-process result”



⍨ Commute

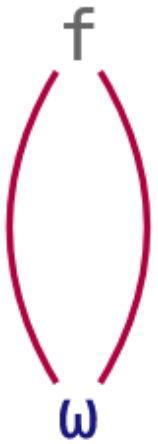
A multiplication table of N is

```
(⍳⍵) ∘.× (⍳⍵)
```

We can write this as

```
∘.×⍨ ⍳⍵
```

“selfie”




```
f⍨ ⍵
```

Tasks: Tacify!

1. `'Hello' {(≠⍺)≡(≠⍵)} 'World'` `1`
2. `'ab' 'cd' 'ab' {+/⍺∈⍵} 'ab' 'de'` `2`
3. `8 100 0 {⍺÷⌊⍵} 2.5` `4 50 0`
4. `10 4 1 0 {⍺⌊⌊⍵} 2.5` `2 2 1 0`
5. `{≠\⍵=⍵} 2 7 1 8 3` `1 0 1 0 1`

Train Introduction

Also a type of composition

Sequence of functions in isolation

- Parenthesised:

```
(+⌿÷≢) 3 1 4 1 5
```

- Assigned

```
Avg←+⌿÷≢
Avg 3 1 4 1 5
```

Most common mistake

```
      +⌿÷≢ 3 1 4 1 5
0.2

     (+⌿÷≢)3 1 4 1 5
2.8
```

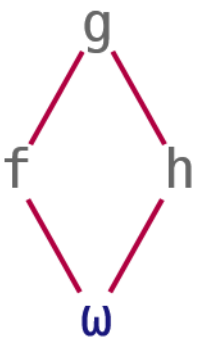
Discussion: Writing Trains

```
(f Y) + (h Y)   →   (f + h) Y
(f Y) + (  Y)   →   (f + ⊢) Y
```

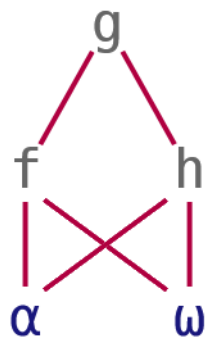
```
(X f Y) + (X h Y)   →   X (f + h) Y
(X    ) + (X h Y)   →   X (⊣ + h) Y
```

Discussion: Writing Trains

```
(f Y) g (h Y)   →   (f g h) Y
(f Y) g (  Y)   →   (f g ⊢) Y
```



(fgh)ω



α(fgh)ω

```
(X f Y) g (X h Y)   →   X (f g h) Y
(X   ) g (X h Y)   →   X (⊣ g h) Y
```

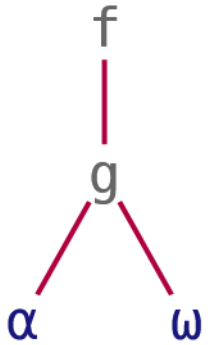
Discussion: Writing Trains

```
   (f Y) g (h Y)   →   (f g h) Y
   (f Y) g (  Y)   →   (f g ⊢) Y
         g (h Y)   →   (  g h) Y

(X f Y) g (X h Y)   →   X (f g h) Y
(X   ) g (X h Y)   →   X (⊣ g h) Y
         g (X h Y)   →   X (  g h) Y
```

Discussion: Writing Trains

```
(f Y) g (h Y)   →   (f g h) Y
(f Y) g (  Y)   →   (f g ⊢) Y
      f (g Y)   →   (  f g) Y

(X f Y) g (X h Y)   →   X (f g h) Y
(X    ) g (X h Y)   →   X (⊣ g h) Y
      f (X g Y)   →   X (  f g) Y
```

(fg)ω



α(fg)ω

Train Details

Some parts can be arrays:

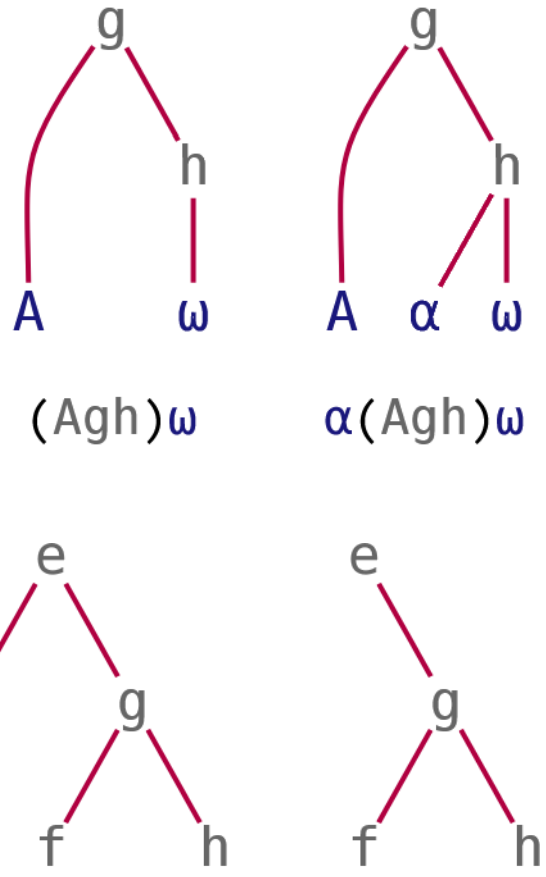
- `A g h` is `{A} g h`

Two sub-types of trains:

- Fork: `f g h` and `A g h`
- Atop: `f g`

Longer trains:

- `(d e f g h)` is `(d e (f g h))`
- `(e f g h)` is `(e (f g h))`



Example: Writing a Train

```
      {∧/ (⌈\⍵) = ⍵}  1 3 5 6 7
1
      {∧/ (⌈\⍵) = (⊢⍵)}  1 3 5 6 7
1
      {∧/ (⌈\ = ⊢) ⍵}  1 3 5 6 7
1
      {(∧/ ⌈\ = ⊢) ⍵}  1 3 5 6 7
1
      (∧/ ⌈\ = ⊢)  1 3 5 6 7
1
```

Tasks: Convert dfn to tacit

1. `{1+⍵} 2 7 1 8`
2. `3 1 4 {(⍺∪⍵)~(⍺∩⍵)} 1 6 1`
3. `{∪⍵∨⍳⍵} 10`
4. `2 {(⍺⊃⍵)⊃⍵} 2 7 1 8 3`
5. `{⍵*2} 2 7 1 8 3`

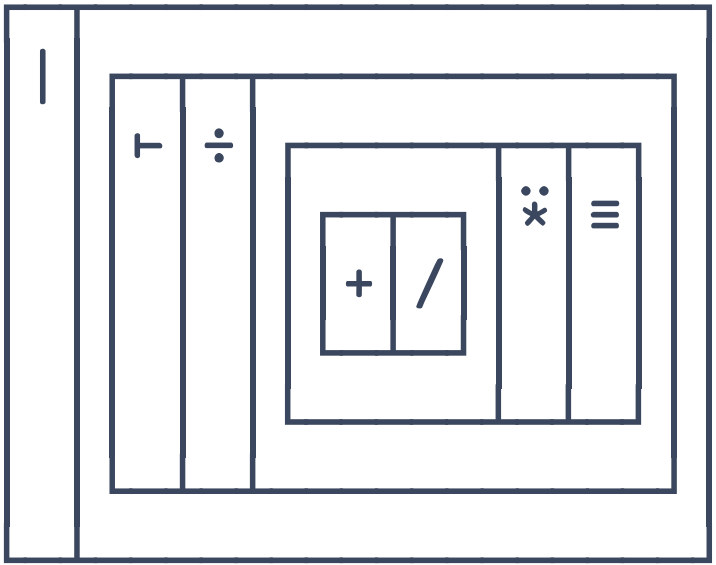
Bonus:
Find 3 ways to do this!

```
3 8 2 9

3 4 6

1 2 5 10

7

4 49 1 64 9
```

Amazing tool: `]box on`

```
]box on
```

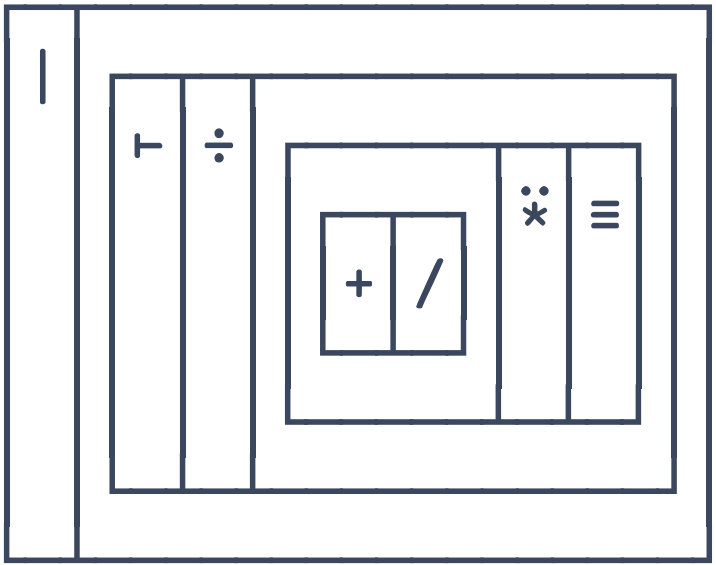
```
|⊢÷+/⍤≡
```



```
]box on -t=…
```

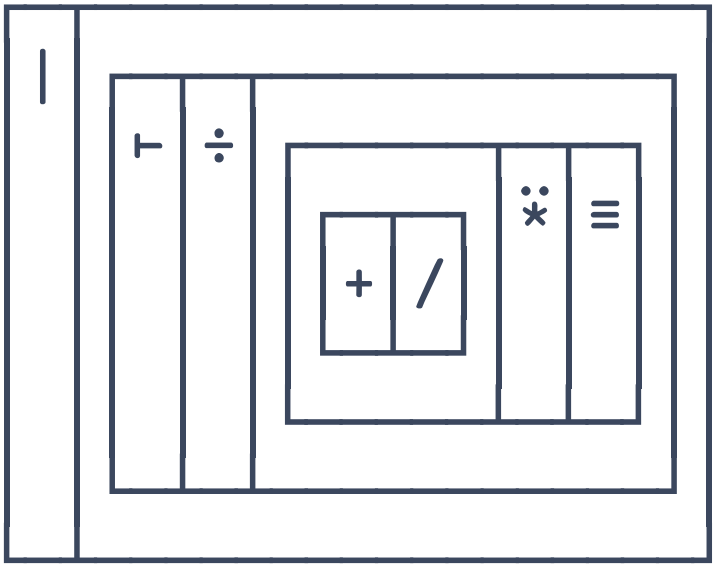
```
|⊢÷+/⍤≡
```

```
-t=box
```

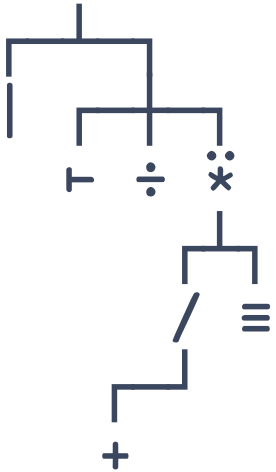


]box on -t=… |⊢÷+/⍤≡

-t=box

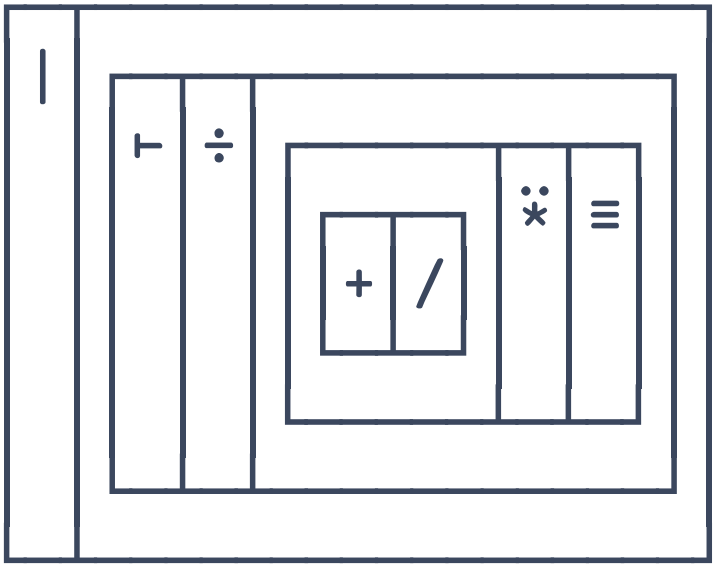


-t=tree

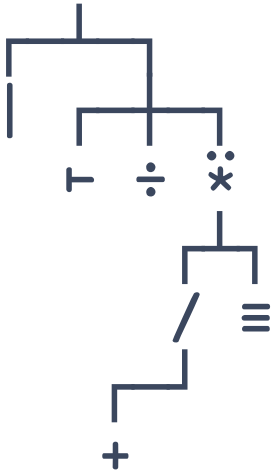


]box on -t=… |⊢÷+/⍤≡

-t=box



-t=tree



-t=parens

```
|(⊢÷((+/)⍤≡))
```

Tasks: Convert dfn to tacit

1. `{(⊖⍵)⊥(⊖⍵)} 1 1 1 0 1 1 0 0` → `3`
2. `',;' {(~⍵∊⍺)⊆⍵} 'ab,de;fgh'` → `ab de fgh`

```
nums←2 7 1 8 3
```

3. `4 {(⍺+≠⍵)÷⍺} nums` → `4.5 4.75`
4. `{(+≠⍵)÷≢⍵} nums` → `4.2`
5. **Bonus task:** Combine 3 & 4 into an ambivalent function.

Issues

- Arguments in operands
- Lots of monadic functions
- Dotting, Assignment, Recursion
- Selection

Issues: Arguments in operands

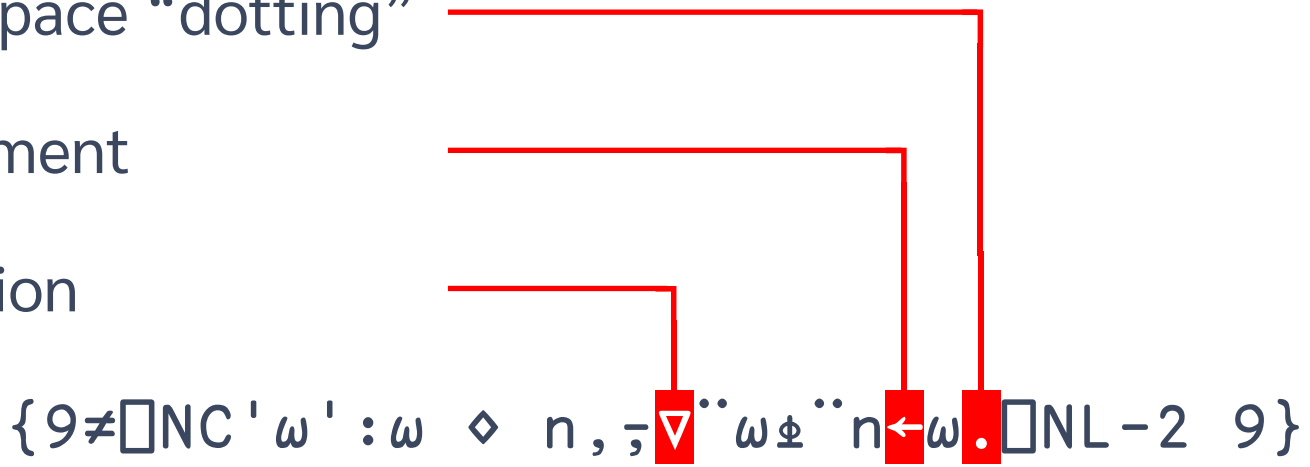
```
      10 {x←⍺ ⋄ ⍵@{x<⍵}⍵} 1 2 13 14 5 16 7 18
1 2 18 16 5 14 7 13
      10 {      ⍵@(⍺<⊢)⍵} 1 2 13 14 5 16 7 18
1 2 18 16 5 14 7 13

      2 {(⍵⍤⌽⍤⍳⍺)⍵} 3 3⍴⍳9
9 8 7
6 5 4
3 2 1
```

Issues: Lots of monadic functions

```
{⍵+≠↑⍵⍨⍵}
⍵⍤(+≠)⍤↑⍤(⍵⍨)
⍵(+≠(↑⍵⍨))
⍵+≠⍤↑⍤(⍵⍨)
⍵(+≠⍤↑⍵⍨)
```

Issues: Recursion, Assignment, Dotting

- Namespace “dotting”
- Assignment
- Recursion



Issues: Selection

```
      {(3>⍵)≠⍵} 3 1 4 1 5
4 5
       (3∘>≠⊢)  3 1 4 1 5
SYNTAX ERROR
       (3∘>≠ ⊢) 3 1 4 1 5
SYNTAX ERROR
     (3∘> ⊢⍤≠ ⊢)3 1 4 1 5
4 5
        3∘>⍤≠   3 1 4 1 5
```

20.0 Conference Edition

Issues: Selection

```
      'aeiou' {ω[α⍋ω]} 'hello world'
eoohll wrld
      'aeiou' {ω[]⍨α⍋ω} 'hello world'
LENGTH ERROR
      'aeiou' {ω[]⍨⊂α⍋ω} 'hello world'
eoohll wrld
      'aeiou'  (⊂⍤⍋[]⊢)  'hello world'
eoohll wrld
      'aeiou'   (⍋⍥⊢)   'hello world'
```

21.0?

Reading Tacit

operator scope

trains

putting it all together

Reading Tacit: operator scope

```
=∘⍤0 1⍨∘⍳5
```

Reading Tacit: operator scope

```
=∘⍤0 1⍨∘⍳5
```

Reading Tacit: operator scope

```
=∘⍤⍥0 1⍨∘⍳5
```

Reading Tacit: operator scope

```
=∘⍤⍥0 1⍨∘⍳5
```

Reading Tacit: operator scope

```
=∘⍋⍤0 1⍨∘⍳5
```

Reading Tacit: operator scope

```
=∘⌽⍤0 1⍨∘⍳5
```

Reading Tacit: operator scope

```
=∘⍤0 1⍨∘⍳5
```

Reading Tacit: operator scope

```
=∘⍤0 1⍨∘⍳5
```

Reading Tacit: operator scope

```
=∘⍋⍤0 1⍨∘⍳5
```

Reading Tacit: operator scope

```
=∘⍤⍥0 1⍨∘⍳5
(((=∘⍤)⍥0 1)⍨)∘⍳)5
```

Reading Tacit: operator scope

```
=∘⍋⍤0 1⍨∘⍳5
```

```
(((=∘⍋)⍤0 1)⍨)∘⍳)5
```

Reading Tacit: operator scope

```
=∘⍧⍤0 1⍨∘⍳5
```

```
(((=∘⍧)⍤0 1)⍨)∘⍳)5
```

Reading Tacit: operator scope

```
=∘⍤0 1⍨∘⍳5
(((=∘⍤)0 1)⍨)∘⍳)5
```

Reading Tacit: operator scope

```
=∘⍋⍤0 1⍨∘⍳5
((((=∘⍋)⍤0 1)⍨)∘⍳)5
```

Writing Tacit: operator scope

```
1  {α⌈+/ω} 3 1 ¯4 1 ¯5
1    ⌈∘+/  3 1 ¯4 1 ¯5
1    ⌈∘+/  3 1 ¯4 1 ¯5
1   (⌈∘+)/ 3 1 ¯4 1 ¯5
1   ⌈∘(+/) 3 1 ¯4 1 ¯5
1  {α⌈+/ω} 3 1 ¯4 1 ¯5
```

Tasks: Convert tacit to dfn

1. Monadic `≠∘+⍨`
2. Monadic `↑⍤⊢,⍤⊂`
3. Dyadic `↑⍤⊢,⍤⊂`
4. Dyadic `+∘÷⍣≡`

Bonus tasks:

5. Combine 2. and 3. into a single ambivalent function
   Hint: use the `{⍺←… ⋄ …}` syntax

Reading Tacit: trains

```
      (+,-) 4
4 ¯4
```

Reading Tacit: trains

```
      ((+4),(-4))
4 ¯4
```

Reading Tacit: trains

```
      (4,¯4)
4 ¯4
```

Reading Tacit: trains

```
      10 (+,-) 4
14 6
```

Reading Tacit: trains

```
      ((10+4),(10-4))
14 6
```

Reading Tacit: trains

```
      (14,6)

14 6
```

Reading Tacit: trains

```
(|+⌿÷1⌈≢) 3 ¯1 ¯4 1 ¯5
```

Reading Tacit: trains

```
(| +⌿ ÷ 1 ⌈ ≢)  3 ¯1 ¯4 1 ¯5
```

Reading Tacit: trains

```
(|   +⌿                    ÷ 1 ⌈   ≢)   3 ¯1 ¯4 1 ¯5
(| (+⌿3 ¯1 ¯4 1 ¯5) ÷ 1 ⌈ (≢3 ¯1 ¯4 1 ¯5))
(|   ¯6                   ÷ 1 ⌈   5                 )
(|   ¯6                   ÷     5                  )
(|                    ¯1.2                        )
                       1.2
```

Reading Tacit: trains

```
      (| +⌿ ÷ 1 ⌈ ≠)  3 ¯1 ¯4 1 ¯5
(| (+⌿3 ¯1 ¯4 1 ¯5) ÷ 1 ⌈ (≠3 ¯1 ¯4 1 ¯5))
          (| ¯6 ÷ 1 ⌈ 5)
            (| ¯6 ÷ 5)
              (| ¯1.2)
                1.2
```

Reading Tacit: putting it all together

```
(⊂⍥⍰⍨⍥≠⍬⊢) 'AEIOU'
```

Reading Tacit: putting it all together

```
(⊂⍤⍰⍨∘≠ ⍬ ⊢)  'AEIOU'
```

Reading Tacit: putting it all together

```
(⊂⍤⍰⍨∘≠                  ⌷  ⊢)  'AEIOU'
((⊂⍤⍰⍨∘≠ 'AEIOU') ⌷ (⊢'AEIOU'))
```

Reading Tacit: putting it all together

```
 (⊂⍤?⍨∘≢           ⎕  ⊢)  'AEIOU'
((⊂⍤?⍨∘≢ 'AEIOU') ⎕ (⊢'AEIOU'))
((⊂⍤?⍨ 5         ) ⎕  'AEIOU'   )
```

Reading Tacit: putting it all together

```
 (  ⊂⍤?⍨∘≢          ⎕  ⊢)  'AEIOU'
((  ⊂⍤?⍨∘≢ 'AEIOU') ⎕ (⊢'AEIOU'))
((  ⊂⍤?⍨ 5        ) ⎕  'AEIOU'  )
((5 ⊂⍤? 5)          ⎕  'AEIOU'  )
```

Reading Tacit: putting it all together

```
 (  ⊂ ⍤ ?⍨∘≠           ) ⌷  ⊢)  'AEIOU'
((  ⊂ ⍤ ?⍨∘≠ 'AEIOU')  ⌷ (⊢'AEIOU'))
((  ⊂ ⍤ ?⍨ 5          ) ⌷  'AEIOU'  )
((5 ⊂ ⍤ ? 5)           ⌷  'AEIOU'  )
((  ⊂ 5 ? 5)           ⌷  'AEIOU'  )
```

Reading Tacit: putting it all together

```
(⊂⍤⍰⍤⍥≠ ⌷ ⊢)  'AEIOU'
((⊂⍤⍰⍤⍥≠ 'AEIOU') ⌷ (⊢'AEIOU'))
((⊂⍤⍰⍤ 5) ⌷ 'AEIOU')
((5 ⊂⍤⍰ 5) ⌷ 'AEIOU')
   ((⊂ 5 ⍰ 5) ⌷ 'AEIOU')
```

Tasks: Convert tacit to dfn

1. Monadic `××⌊⍤⍥|`
2. Dyadic `⌊∘≢↑⊢`
3. Dyadic `≡⍤(⍞C~∘' ')`
4. Monadic `+/⊢>+/÷≢`

Bonus task:

5. Monadic `⍬≡⊢⊢≡⍬`

Tasks: Determine valence

1. `1+⍴⊥⍣¯1∘+`
2. `+/∧\⍤=`
3. `|⍣¯11|1⊥+\`
4. `1 1⍱∘.+`
5. `,⍤⍱`
6. `/⍤,`

Reading Tacit: tacit.help

Transform tacit APL into dfn form

```
      f ← +⌿÷⊣∘≢
      f Y ⇔ {((+⌿)ω)÷(≢ω)}
X     f Y ⇔ {(α(+⌿)ω)÷α}
```

Arrays: $A, B, C, \ldots$ Functions: $a, b, c, \ldots$

Task: Working with tacit code

This dyadic function is like ⍳ but gives 0 for “not found”:

(⍳ ×≠⍤⍥⊣≥ ⍳)

Make it more efficient by:

- Breaking out ⍳ from the parenthesis
  Hint: Pass all the data you need to the inner function

Then simplify it by:

- Using ⍤⍥ instead of ∘ and adjusting the rest as necessary

Task: Working with tacit code

This function is applied to a non-empty numeric vector:

```
(+\∊⍨∘⍳+/)
```

Make it more efficient by:

- Breaking out `+\` from the parenthesis
- Computing what would be `+/` from `+\`
  Hint: `+/` is the last element of `+\`

Function composition:

- ⍤⍥ *Pre-process both*
- ⍤ *Pre-process right*
- ⍥ *Post-process*
- ⍨ *Selfie*

Operators: long left scope

Trains: odd-even from right

Tools: `]box on -t=…`
tacit.help

Issues with tacit:

Arguments in operands
Lots of monadic functions
Namespace “dotting”
Assignment
Recursion
Selection:

```
⊢⍥≠
⊂⍥… ⍬ …
```