DYALOG

APL Problem Solving Competition Phase 1

Introduction

The Phase 1 problems are designed to be solved using short APL functions. If you find yourself writing more than a couple of statements in your solution, then there is probably a better way to do it.

Submission format

Each solution must be a single dfn or tacit function.

A dfn is one or more APL statements enclosed in braces `{}`. The left hand argument, if any, is represented in a dfn by `⍺`, while the right hand argument is represented by `⍵`. For example:

```
      'Hello' {⍺,'-',⍵,'!'} 'world'
Hello-world!
```

A dfn terminates on the first statement that is not an assignment. If that statement produces a value, then the dfn returns that value as its result. The diamond symbol `⋄` separates APL statements. For example:

```
      'left' { ⍵ ⋄ ⍺ } 'right'
right
```

More information on dfns can be found on the APL Wiki.

A tacit function is an APL expression that does not explicitly mention its arguments. In the example below, `(+⌿÷≢)` is a tacit function that computes the average of a vector (list) of numbers:

```
      (+⌿÷≢) 1 2 3 4 5 6
3.5
```

More information on tacit functions can be found on the APL Wiki.

Judging Guidelines

Phase 1 will mainly be judged based on:

- Generality: does your function handle the given basic and edge cases?
- Use of array-oriented thinking: did you write array-oriented APL or something that looks more like C# written in APL?

You should not include comments in your Phase 1 solutions.

Tips

- Several of the problem descriptions will describe arguments that can be a scalar (a single element) or a vector (a list). This is largely pedantic, but in such cases your functions should produce correct results for both types of input.
- The symbol `⍝` is the APL comment symbol. In some of the examples, we provide comments to give you more information about that particular example.
- Some of the problem test cases use "boxed display" to make the structure of the returned results clearer. Boxing is always active on TryAPL and can be enabled in your local APL Session with the `]Box` user command:

```
      ⍳¨⍳4
 1  1 2  1 2 3  1 2 3 4
      ]Box on
Was OFF
      ⍳¨⍳4
┌─┬───┬─────┬───────┐
│1│1 2│1 2 3│1 2 3 4│
└─┴───┴─────┴───────┘
```

Sample problem

The content of the orange box shows what a typical Phase 1 problem description looks like. It also presents some possible solutions of varying quality, and explains how to provide your own solution.

Each problem starts with a task description; some also include a hint suggesting one or more APL primitives. These may be helpful in solving the problem, but you are under no obligation to use them. Clicking on a primitive in the hint opens the Dyalog documentation page for that primitive.

Each problem ends with some example cases. You can use these as a basis for implementing your solution.

Counting Vowels A

Write an APL function to count the number of vowels (A, E, I, O, U) in an array consisting of uppercase letters (A–Z).

Hint: The membership function `X∊Y` could be helpful for this problem.

Examples

```
      (fn) 'COOLAPL'
3
      (fn) ''    ⍝ empty argument
0
      (fn) 'NVWLSHR'    ⍝ no vowels here
0
```

Below are three sample solutions. All three produce the correct answer, but the first two functions would be ranked higher by the competition judging committee as they demonstrate better use of array-oriented programming than the third one.

```
      ({+/⍵∊'AEIOU'}) 'COOLAPL'    ⍝ good dfn
3
      (+/∊∘'AEIOU') 'COOLAPL'    ⍝ good tacit function
3
      ⍝ suboptimal dfn:
      {(+/⍵='A')+(+/⍵='E')+(+/⍵='I')+(+/⍵='O')+(+/⍵='U')} 'COOLAPL'
3
```

1: Counting DNA Nucleotides 🧬

This problem was inspired by Counting DNA Nucleotides found on the excellent bioinformatics website rosalind.info.

Write a function that:

- takes a right argument that is a character vector or scalar representing a DNA string (whose alphabet contains the symbols 'A', 'C', 'G', and 'T').
- returns a 4-element numeric vector containing the counts of each symbol 'A', 'C', 'G', and 'T' respectively.

💡 **Hint:** The *key* operator `f⌸` or the *outer product* operator `∘.g` could be helpful.

Examples

```
      (fn)
'AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC'
20 12 17 21

      (fn) ''
0 0 0 0

      (fn) 'G'
0 0 1 0
```

2: Attack of the Mutations! ≠

This problem is inspired by the Counting Point Mutations problem found on the excellent Bioinformatics education website rosalind.info.

Write a function that:

- takes right and left arguments that are character vectors or scalars of equal length – these represent DNA strings.
- returns an integer representing the Hamming distance (the number of differences in corresponding positions) between the arguments.

Hint: The *plus* function `X+Y` could be helpful.

Examples

```
      'GAGCCTACTAACGGGAT' (fn) 'CATCGTAATGACGGCCT'
7

      'A' (fn) 'T'
1

      '' (fn) ''
0

      (fn)⍨ 'CATCGTAATGACGGCCT'
0
```

3: Uniquely Qualified

Write a function that:

- takes right and left arguments that are arrays of arbitrary rank, depth, and value.
- returns a vector of all elements that appear in either of the two argument arrays but not in both. The order of elements in the result is not significant.

Hint: The *without* function `X~Y` could be helpful.

Examples

```
      'DYALOG' (fn) 'APL'
DYOGP

      'DYALOG'  (fn) ⊂'APL'
┌─┬─┬─┬─┬─┬─┬───┐
│D│Y│A│L│O│G│APL│
└─┴─┴─┴─┴─┴─┴───┘

      (2 2⍴'Hello'(⊂'World')(2 2⍴⍳4)42) (fn) 42 'Have a nice day'
┌─────┬───────┬───┬───────────────┐
│Hello│┌─────┐│1 2│Have a nice day│
│     ││World││3 4│               │
│     │└─────┘│   │               │
└─────┴───────┴───┴───────────────┘

      1 1 1 (fn) 2 2
1 1 1 2 2
```

4: In the Long One...

Write a function that:

- takes a right argument that is a Boolean scalar or vector.
- returns the length of the longest sequence of consecutive 1s.

Hint: The *partition* function `X⊆fY` could be helpful.

Examples

```
      (fn) 1 1 1 0 1 1 0 0 1 1 1 1 0
4
      
      (fn) ⍬
0

      (fn) 1
1

      (fn) 0
0

      (fn) 12/0 1 0 1
12
```

5: Stairway to Heaven (with apologies to Led Zeppelin)

Write APL function that:

- Given a scalar integer argument, `n`, in the range 0-100.
- Returns a character matrix comprised of spaces and `⎕` that resembles an `n`-level left-to-right ascending stairway.

Hint: The *index generator* function `⍳Y` could help with solving this problem.

Examples

```
      (fn) 10
         ⎕
        ⎕⎕
       ⎕⎕⎕
      ⎕⎕⎕⎕
     ⎕⎕⎕⎕⎕
    ⎕⎕⎕⎕⎕⎕
   ⎕⎕⎕⎕⎕⎕⎕
  ⎕⎕⎕⎕⎕⎕⎕⎕
 ⎕⎕⎕⎕⎕⎕⎕⎕⎕
⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕

      (fn) 0 ⍝ returns a 0×0 matrix
```

6: Pyramid Scheme

Write a monadic function that:

- takes an argument *n* that is an integer scalar in the range 0-100.
- returns a square matrix "pyramid" with `0⌈¯1+2×n` rows and columns of `n` increasing concentric levels.
  By this we mean that the center element of the matrix will be `n`, surrounded on all sides by `n-1`.

Hint: The functions *minimum* `X⌊Y` and *reverse* `⌽Y`, and the *outer product* operator `X∘.gY` could be helpful.

Examples

```
      (fn) 3
1 1 1 1 1
1 2 2 2 1
1 2 3 2 1
1 2 2 2 1
1 1 1 1 1

      (fn) 5
1 1 1 1 1 1 1 1 1
1 2 2 2 2 2 2 2 1
1 2 3 3 3 3 3 2 1
1 2 3 4 4 4 3 2 1
1 2 3 4 5 4 3 2 1
1 2 3 4 4 4 3 2 1
1 2 3 3 3 3 3 2 1
1 2 2 2 2 2 2 2 1
1 1 1 1 1 1 1 1 1

      (fn) 1 ⍝ should return 1 1⍴1
1

      (fn) 0 ⍝ should return 0 0⍴0
```

7: Just Golfing Around

Apologies to the code golfers out there, but this problem has nothing to do with code golf! Instead, it addresses the problem of assigning places in a golf tournament. In regular golf, lower scores place higher – the lowest score places first and the highest score places last.

Write a function that:

- takes a right argument that is a non-decreasing vector or scalar of strictly positive integers, representing a set of scores.
- returns a numeric vector of the place for each score; for duplicate scores, it returns the average of the places they hold.

Hint: This problem has several viable approaches including using *key* `f⌸`, or *partition* `X⊆Y`, or *interval index* `X⍸Y`.

Examples

```
      (fn) 1 2 3 4 5
1 2 3 4 5

      (fn) 68 71 71 73
1 2.5 2.5 4

      (fn) 67 68 68 69 70 70 70 71 72
1 2.5 2.5 4 6 6 6 8 9

      (fn) 6⍴70
3.5 3.5 3.5 3.5 3.5 3.5

      (fn) ⍬ ⍝ this should return an empty vector


      (fn) 67 ⍝ should work with a scalar argument
1
```

8: Let's Split!

Write a function that:

- takes a right argument that is a non-empty character vector or scalar.
- takes a left argument that is a non-empty character vector or scalar.
- returns a 2-element vector of character vectors in which the right argument is split immediately before the *first* occurence of *any* element in the left argument. If no left-argument element occurs in the right argument, then the split should happen after the last element of the right argument.

Hint: The *take* `X↑Y` and *drop* `X↓Y` functions, or the *partitioned enclose* function `X⊆Y`, could be helpful.

Examples

```
      'do' (fn) 'Hello World'
┌────┬───────┐
│Hell│o World│
└────┴───────┘

      'KEI' (fn) ⎕A ⍝ ⎕A is the system constant that contains the
characters A-Z
┌────┬──────────────────────┐
│ABCD│EFGHIJKLMNOPQRSTUVWXYZ│
└────┴──────────────────────┘

      (⌽⎕A) (fn) ⎕A
┌┬──────────────────────────┐
││ABCDEFGHIJKLMNOPQRSTUVWXYZ│
└┴──────────────────────────┘

      ⎕D (fn) ⎕A ⍝ ⎕D is the system constant that contains the characters
0-9
┌──────────────────────────┬┐
│ABCDEFGHIJKLMNOPQRSTUVWXYZ││
└──────────────────────────┴┘

      ⎕D (fn) 'Q'
┌─┬┐
│Q││
└─┴┘
      ⎕A (fn) 'Q'
┌┬─┐
││Q│
└┴─┘
```

9: An Average Window (or a Windowed Average) 🪟

Write a function that:

- takes a right argument `Y` that is a numeric scalar or non-empty vector.
- takes a left argument `X` that represents the number of neighboring elements on either side of each element in `Y`.
- returns a numeric vector or scalar where each element is the average (mean) of the corresponding element in `Y` and its `X` neighbors on either side. If an element has fewer than `X` neighbors on either side, replicate the first and last values as necessary to make `X` neighbors.

💡 **Hint:** The *Reduce N-Wise* operator `Xf/Y` could help with solving this problem.

Examples

```
      0 (fn) 1 2 3 4 5 6 ⍝ 0 neighbors on each side
1 2 3 4 5 6

      1 (fn) 1 2 3 4 5 6 ⍝ 1 neighbors on each side
1.333333333 2 3 4 5 5.666666667

      2 (fn) 1 2 3 4 5 6 ⍝ 2 neighbors on each side
1.6 2.2 3 4 4.8 5.4

      6 (fn) 1 2 3 4 5 6
2.538461538 2.923076923 3.307692308 3.692307692 4.076923077 4.461538462

      10 (fn) 42
42
```

10: Separation Anxiety

Write a function that:

- takes a right argument that is a character vector or scalar representing a valid non-negative integer.
- takes a left argument that is a character scalar "separator" character.
- returns a character vector that is a representation of the right argument formatted such that the separator character is found between trailing groups of 3 digits.

Note that the number of digits in the character representation might exceed the number of digits that can be represented as a 32-bit integer.

Hint: The *at* operator `@` could be helpful.

Examples

```
      ',' (fn)¨'1' '10' '100' '1000' '10000' '100000' '1000000' 
'10000000' '100000000' '1000000000' '10000000000'
┌─┬──┬───┬─────┬──────┬───────┬─────────┬──────────┬───────────┬──────────
│1│10│100│1,000│10,000│100,000│1,000,000│10,000,000│100,000,000│1,000,000,0
└─┴──┴───┴─────┴──────┴───────┴─────────┴──────────┴───────────┴──────────


      '.' (fn) 60⍴⌽⎕D
987.654.321.098.765.432.109.876.543.210.987.654.321.098.765.432.109.876.543


      '/' (fn) ,'9' ⍝ scalars and 1-element character vectors are 
equivalent
9
```