2019

DYALOG

APL Problem Solving Competition Phase I

The following shows what a typical Phase I problem description looks like. It also presents some possible solutions of varying quality, and explains how to provide your own solution. Content that doesn't appear in regular Phase I problems is formatted like this paragraph.

Each problem begins with a task description, followed by 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 will open the Dyalog documentation page for the suggested primitive.

After the hint is a section of example cases which, among others, are included in the automated tests. Use these as a basis for implementing your solution.

Sample: Counting Vowels

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 ∊ 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. This is because the first two demonstrate better use of array-oriented programming.

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

Phase I Problem Set

1: Chunky Monkey 🐒

Write a function that, given a scalar or vector as the right argument and a positive (>0) integer chunk size **n** as the left argument, breaks the array's items up into chunks of size **n**. If the number of elements in the array is not evenly divisible by **n**, then the last chunk will have fewer than **n** elements.

💡 **Hint:** The partitioned enclose function `⊂` could be helpful for this problem.

Examples

```
      3 (fn) ⍳9    ⍝ ]Box on is used to display the result
┌─────┬─────┬─────┐
│1 2 3│4 5 6│7 8 9│
└─────┴─────┴─────┘

      3 (fn) ⍳11
┌─────┬─────┬─────┬─────┐
│1 2 3│4 5 6│7 8 9│10 11│
└─────┴─────┴─────┴─────┘

      10 (fn) 'Dyalog'
┌──────┐
│Dyalog│
└──────┘

      2 (fn) 'The' 'cat' 'in' 'the' 'hat' 'sat' 'pat'
┌─────────┬────────┬─────────┬─────┐
│┌───┬───┐│┌──┬───┐│┌───┬───┐│┌───┐│
││The│cat│││in│the│││hat│sat│││pat││
│└───┴───┘│└──┴───┘│└───┴───┘│└───┘│
└─────────┴────────┴─────────┴─────┘

      5 (fn) ''    ⍝ result is 0-element vector of text vectors

      4 (fn) 5
┌─┐
│5│
└─┘
```

2: Making the Grade 🎓

Score Range	Letter Grade
0–64	F
65–69	D
70–79	C
80–89	B
90–100	A

Write a function that, given an array of integer test scores in the inclusive range 0–100, returns an identically-shaped array of the corresponding letter grades according to the table to the left.

💡 **Hint:** You may want to investigate the interval index function ⍸.

Examples

```
      (fn) 0 64 65 69 70 79 80 89 90 100
FFDDCCBBAA

      (fn) ⍬  ⍝ returns an empty vector


      (fn) 2 3⍴71 82 81 82 84 59
CBB
BBF
```

3: Grade Distribution

The school's administrative department wants to publish some simple statistics. Given a non-empty character vector of single-letter grades, produce a 3-column, 5-row, alphabetically-sorted matrix of each grade, the number of occurrences of that grade, and the percentage (rounded to 1 decimal position) of the total number of occurrences of that grade. The table should have a row for each grade even if there are no occurrences of a grade. Note: due to rounding the last column might not total 100%.

Hint: The key operator ⌸ could be useful for this problem.

Examples

```
      (fn) 9 3 8 4 7/'DABFC'
A 3  9.7
B 8 25.8
C 7 22.6
D 9 29
F 4 12.9

      (fn) 20⍴'ABC'
A 7 35
B 7 35
C 6 30
D 0  0
F 0  0

      (fn) ,'B'
A 0   0
B 1 100
C 0   0
D 0   0
F 0   0
```

4: Knight Moves

1 1	1 2	1 3	1 4	1 5	1 6	1 7	1 8
2 1	2 2	2 3	2 4	2 5	2 6	2 7	2 8
3 1	3 2	***3 3***	3 4	***3 5***	3 6	3 7	3 8
4 1	***4 2***	4 3	4 4	4 5	***4 6***	4 7	4 8
5 1	5 2	5 3	**5 4**	5 5	5 6	5 7	5 8
6 1	***6 2***	6 3	6 4	6 5	***6 6***	6 7	6 8
7 1	7 2	***7 3***	7 4	***7 5***	7 6	7 7	7 8
8 1	8 2	8 3	8 4	8 5	8 6	8 7	8 8

Consider a chess board as an 8×8 matrix with square `(1 1)` in the upper left corner and square `(8 8)` in the lower right corner. For those not familiar with the game a chess, the knight, generally depicted as a horse (♞), can move 2 spaces right or left and then 1 space up or down, or 2 spaces up or down and then 1 space right or left. This means that a **knight** on square `(5 4)` can move to any of the ***indicated*** squares.

Given a 2-element vector representing the current square for a knight, return a vector of 2-element vectors representing (in any order) all the squares that the knight can move to.

Hint: The outer product operator `∘.` could be useful for generating the coordinates.

Examples

```
      (fn) 5 4    ⍝ ]Box on is used to display the result
┌───┬───┬───┬───┬───┬───┬───┬───┐
│3 3│3 5│4 2│4 6│6 2│6 6│7 3│7 5│
└───┴───┴───┴───┴───┴───┴───┴───┘

      (fn) 1 1
┌───┬───┐
│2 3│3 2│
└───┴───┘
```

5: Doubling Up

Given a word or a list of words, return a Boolean vector where `1` indicates a word with one or more consecutive duplicated, case-sensitive, letters. Each word will have at least one letter and will consist entirely of either uppercase (A-Z) or lowercase (a-z) letters. Words consisting of a single letter can be scalars.

Hint: The nest function `⊆` could be useful.

Examples

```
      (fn) 'I' 'feed' 'the' 'bookkeeper'
0 1 0 1

      (fn) 'I'
0

      (fn) 'feed'
1

      (fn) 'MY' 'LLAMAS' 'HAVE' 'BEEN' 'GOOD'
0 1 0 1 1
```

6: Telephone Names

Some telephone keypads have letters of the alphabet embossed on their keytops. Some people like to remember phone numbers by converting them to an alphanumeric form using one of the letters on the corresponding key. For example, in the keypad shown, `'ALSMITH'` would correspond to the number 257-6484 and `'1DYALOGBEST'` would correspond to 1-392-564-2378.

Write an APL function that takes a character vector right argument that consists of digits and uppercase letters and returns an integer vector of the corresponding digits on the keypad.

Hint: Your solution might make use of the membership function `∊` .

Examples

```
      (fn) 'IAMYY4U'
4 2 6 9 9 4 8

      (fn) ''   ⍝ should return an empty vector


      (fn) 'UR2CUTE'
8 7 2 2 8 8 3
```

7: In the Center of It All

Given a right argument of a list of words (or possibly a single word) and a left argument of a width, return a character matrix that has width columns and one row per word, with each word is centered within the row. If width is smaller than the length of a word, truncate the word from the right. If there are an odd number of spaces to center within, leave the extra space on the right.

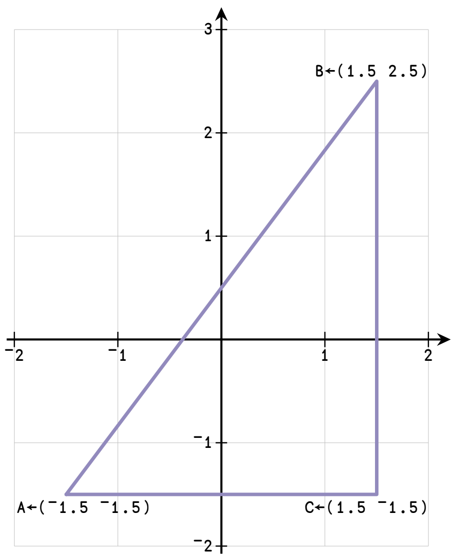
Hint: The mix ↑ and rotate ⌽ functions will probably be useful here.

Examples

```
      10 (fn) 'APL' 'Problem' 'Solving' 'Competition'
   APL
 Problem
 Solving
Competitio

      3 (fn) 0⍴⊂''    ⍝ result should be 0-row, 3-column matrix
```

8: Going the Distance 🥾



Given a vector of `(X Y)` points, or a single `X Y` point, determine the total distance covered when travelling in a straight line from the first point to the next one, and so on until the last point, then returning directly back to the start.

For example, given the points `(A B C) ← (¯1.5 ¯1.5) (1.5 2.5) (1.5 ¯1.5)`, the distance `A` to `B` is `5`, `B` to `C` is `4` and `C` back to `A` is `3`, for a total of `12`.

💡 **Hint:** The rotate `⌽` and power `*` functions might be useful.

Examples

```
      (fn) (1 ¯1)(1 3)    ⍝ from A to B and back to A
8

      (fn) (1 1)(1 2)(2 2)(2 1)    ⍝ from A to B to C to D to A
4

      (fn) 5 5    ⍝ staying where we are
0

      (fn) (1 1)(3 3)    ⍝ there and back again
5.656854249
```

9: Area Code à la Gauss

Gauss's area formula, also known as the shoelace formula, is an algorithm to calculate the area of a simple polygon (a polygon that does not intersect itself). It's called the shoelace formula because of a common method using matrices to evaluate it.

For example, the area of the triangle described by the vertices `(2 4)(3 ¯8)(1 2)` can be calculated by "walking around" the perimeter back to the first vertex, then drawing diagonals between the columns as shown below. The pattern created by the intersecting diagonals resembles shoelaces, hence the name "shoelace formula".

Hint: You may want to investigate the rotate first `⊖` function.

First place the vertices in order above each other:	2 4 3 ¯8 1 2 2 4

Sum the products of the numbers connected by the diagonal lines going down and to the right:
`      (2×¯8)+(3×2)+(1×4)`
`¯6`
	2 4
	3 ¯8
	1 2
	2 4

Next sum the products of the numbers connected by the diagonal lines going down and to the left:
`      (4×3)+(¯8×1)+(2×2)`
`8`
	2 4
	3 ¯8
	1 2
	2 4

Finally, halve the absolute value of the difference between the two sums:
`      0.5 × | ¯6 - 8`
`7`
	2 4
	3 ¯8
	1 2
	2 4

Given a vector of `(X Y)` points, or a single `X Y` point, return a number indicating the area circumscribed by the points.

Examples

```
      (fn) (2 4)(3 ¯8)(1 2)
7

      (fn) (1 1)   ⍝ a point has no area
0

      (fn) (1 1)(2 2)   ⍝ neither does a line
0
```

10: Odds & Evens

Given a vector of words, separate the words into two vectors – one containing all the words that have an odd number of letters and the other containing all the words that have an even number of letters.

Hint: You may want to look into the dyadic form of the key operator ⌸.

Examples

```
      (fn) 'the' 'plan' 'is' 'great'   ⍝ ]box on is used to display the
result
┌───────────┬─────────┐
│┌───┬─────┐│┌────┬──┐│
││the│great│││plan│is││
│└───┴─────┘│└────┴──┘│
└───────────┴─────────┘

      (fn) 'all' 'odd' ⍝ note the empty 2nd element of the result
┌─────────┬┐
│┌───┬───┐││
││all│odd│││
│└───┴───┘││
└─────────┴┘

      (fn) 'only' 'even' 'here' ⍝ note the empty 1st element of the result
┌┬────────────────┐
││┌────┬────┬────┐│
│││only│even│here││
││└────┴────┴────┘│
└┴────────────────┘
```