Task 1

Begin with a clear workspace:

```
)CLEAR
```

Save your workspace on the file system:

```
)SAVE C:\path\to\exercises2_your_name.dws
```

Task 2

Re-write the following single-line dfn as a multi-line dfn to eliminate the in-line assignment:

```
Bayes←{prod÷+/prod←α×ω}
```

Task 3

Make this dfn into the equivalent tradfn:

```
OR←{ ⍝ Probabilistic OR
     p←1-α
     q←1-ω
     1-p×q
}
```

Task 4

Put the following tradfn in your workspace:

```
      ∇ r←Anagram b;Norm
[1]     r←(Norm a)≡(Norm b)
      ∇
```

Put the following dfn in your workspace:

```
   Norm←{ω[⍋ω]~' '}
```

The `Anagram` function has **two** bugs preventing it from working. Fix them so the following expressions work:

```
      'ELEVEN PLUS TWO' Anagram 'TWELVE PLUS ONE'
1
      'ELEVEN PLUS TWO' Anagram 'TEN PLUS THREE'
0
```

Task 5

Write a function `Square` which takes a simple character vector argument and creates a 2-row, 2-column matrix where every element is the given text:

```
      ]disp Square 'abc'
┌→──────┬──────┐
↓abc    │abc   │
├──────→┼──────→┤
│abc    │abc   │
└──────→┴──────→┘
      ]disp Square 'mytext'
┌→────────┬────────┐
↓mytext   │mytext  │
├────────→┼────────→┤
│mytext   │mytext  │
└────────→┴────────→┘
```

Task 6

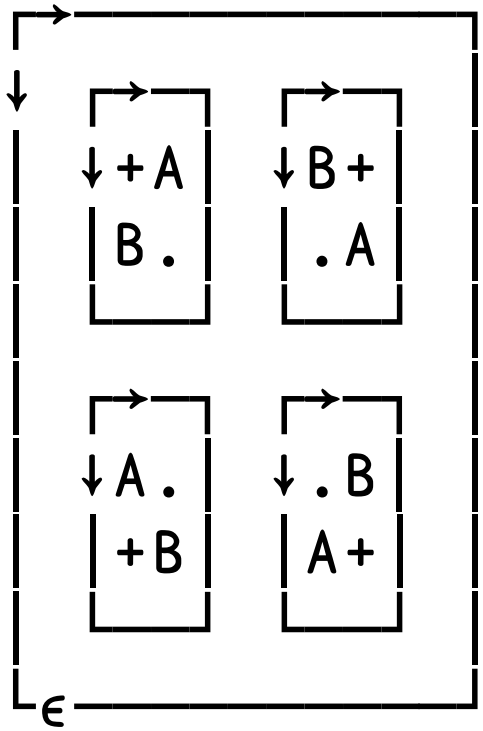
Write a function `Backwards` which takes a vector of vectors and reverses both the overall vector and its elements:

```
      ]disp Backwards 'uoY' 'erA' 'woH' 'olleH'
┌→──────┬─────┬─────┬─────┐
│Hello  │How  │Are  │You  │
└───────┴─────┴─────┴─────┘
      ]disp Backwards (1 2 3)(4 5)(8 2 10)
┌→────────┬───┬─────┐
│10 2 8   │5 4│3 2 1│
└~────────┴~──┴~────┘
```

Task 7

Write a function `Corners` which takes a matrix and returns a 2-by-2 matrix of the 4 rotations of the given matrix:

```
      Corners 2 2⍴'+AB.'
```



Task 8

Create the following variable:

```
]disp table
┌→──────┬──────┬──────┐
↓Air    │Boat  │Car   │
├──────→┼──────→┼──────→┤
│ 1    │ 2    │ 3    │
├~─────┼~─────┼~─────┤
│ 4    │ 5    │ 6    │
├~─────┼~─────┼~─────┤
│ 7    │ 8    │ 9    │
└~─────┴~─────┴~─────┘
```

Task 9

Write a function `From` which takes a character vector left argument and matrix right argument. It must return the first column of the matrix where the top element of that column matches the left argument vector, but without the heading:

```
      'Boat' From table
2 5 8
      'Car' From table
3 6 9
      'B' From (3 3⍴'ABC',3 5 1 3 4 2)
5 4
```

Task 10

Create a variable `nest` which has the following properties:

```
      ⍴nest
2 3
      ≡nest
¯2
      ⍴¨nest
┌──┬──┐
│  │ 2│
├──┼──┤
│3 │ 6│
└──┴──┘
      ]display ∊nest
┌→──────────────────────┐
│I 3 am 1 5 8 amatrix│
└+──────────────────────┘

      ⍴∊nest
14
```

Task 11

Create a function `ReplaceHead` which returns its left argument vector α, but with the first ρω elements replaced with the contents of ω:

```
      'apple' ReplaceHead 'Eat'
Eatle
      'apple' ReplaceHead 'rang'
range
      'apple' ReplaceHead 'ENTERPRISE'
ENTER
```

Task 12

Create a function `SplitOnFirst` which takes a character scalar left argument and simple character vector right argument. It returns a nested vector of character vectors. The first element of the result contains ω until (and not including) the first appearance of α in ω. The second element contains the rest of ω.

```
      ','SplitOnFirst 'this,text'
┌────┬────┐
│this│text│
└────┴────┘
      '|'SplitOnFirst 'split|the|first'
┌─────┬──────────┐
│split│the|first│
└─────┴──────────┘
      ' 'SplitOnFirst 'head and then the tail'
┌────┬────────────────────┐
│head│and then the tail│
└────┴────────────────────┘
```

Task 13

Create a function `ReplaceRow` which returns an array the same shape as its left argument, except that the row specified in the 1st element of its right argument is replaced with the 2nd element. If the 2nd element has the correct length, it is distributed throughout the row.

```
      ]disp (2 3⍴⍳12) ReplaceRow 2 ('yo')
┌→──┬──┬──┐
↓1  │2 │3 │
├~──┼~─┼~─┤
│yo │yo│yo│
└──→┴──→┴──→┘
      ]disp (4 3⍴⍳12) ReplaceRow 2 ('you')
 1  2  3
 y  o  u
 7  8  9
10 11 12
      ]disp (2 3⍴⍳12) ReplaceRow 2 (⍳4)
┌→──────────┬──────────┬──────────┐
↓1          │2         │3         │
├~──────────┼~─────────┼~─────────┤
│1 2 3 4    │1 2 3 4   │1 2 3 4   │
└~─────────→┴~────────→┴~────────→┘
      ]disp (2 3⍴⍳12) ReplaceRow 2 ((1 1)(5 5)(9 9))
┌→────┬────┬────┐
↓1    │2   │3   │
├~───┼~───┼~───┤
│1 1 │5 5 │9 9 │
└~──→┴~──→┴~──→┘
```

Task 14

Create a function `Get` which takes a 2-element nested vector of character vectors as its right argument. It should return the value of the name `ω[2]` in the namespace specified by `ω[1]`.

```
      ns←⎕ns ⍬
      ns.var←1 2 3
      Get 'ns' 'var'
1 2 3

      ns2←⎕ns ⍬
      ns2.got←'my variable'
      Get 'ns2' 'got'
my variable
```

Task 15

Create a function `Reveal` which:

- Takes a namespace reference argument. The namespace will contain two member variables: `password` and `codenum`
- Modifies the contents of the namespace as a side-effect, creating the member variable `codeword`
- Returns the character vector `'Done'` as its result

The member variable `password` is a numeric vector. The member variable `codenum` is a numeric scalar.

The `codenum` is the number of steps to rotate ⎕A for a rotation cypher. The password is indices into the rotated alphabet. For example, if code is 5, then `22 11 7` becomes `'APL'`.

```
      (5⌽⎕A)[22 11 7]
APL
```

If we set up our secret namespace, then Reveal should work as follows:

```
      s ← ⎕ns⍬
      s.password ← 22 11 7
      s.codenum ← 5
      Reveal s
Done
      s.codeword
APL

      secret ← ⎕ns⍬
      secret.password ← 6 18 16 5 18 7 6 14 8 16 18
      secret.codenum ← 13
      Reveal secret
Done
      secret.codeword
SECRETSAUCE
```

Task 16

Write a function `RotationOf` which takes two vectors and determines if they are rotations of each other:

```
      'carrace' RotationOf 'racecar'
1
      'teapot' RotationOf 'topeat'
0
      'apple' RotationOf 'leapp'
1
      'pepper' RotationOf 'repppe'
0
```