tasks (/github/abrudz/tasks/tree/main)
 / Tasks8 Answers.ipynb (/github/abrudz/tasks/tree/main/Tasks8 Answers.ipynb)

Partitioning

APL has two similar functions for partitioning arrays into nested arrays:

Partitioned-enclose `α⊂ω` and partition `α⊆ω`

Migration Level

The behaviour of the *partition* function can be achieved in 2 ways:

Prior to Dyalog version 16.0, use `Partition←{⎕ML←3 ⋄ α⊂ω}`

From version 16.0 onwards, use `Partition←{α ⎕U2286 ω}`

The monadic function **enclose-if-simple** can be encoded as a dfn: `EIS ← {1≥≡ω: ⊂ω ⋄ ω}`

From version 16.0 onwards, use `EIS←{⎕U2286 ω}`

The only relation between partition, partitioned-enclose and enclose-if-simple is the use of left-shoe and left-shoe-underbar symbols, and use with nested arrays. Otherwise they are completely different functions.

Enclose Rows

Here is an example of the "enclose last axis" function (tasks8 #14). This is the same as the "split" function (monadic `↓`). Note the difference between "conventional" programming logic in the first solution (traditional function) and "array-oriented" logic in the second solution (dfn).

In [9]:

```
∇  array←EncloseRows array;rank
    rank←⍴⍴array
    :If rank>1
        array←⊂[rank]array
    :Else
        array←⊂array
    :EndIf
∇
```

In [8]:

```
]disp EncloseRows 1 2 3
```

Out[8]:

```
┌──────┐
```

```
│1 2 3│
└~────→┘
```

In [3]:

```
]disp {⊂[(0<r)/r←⍴⍴⍵]⍵} 1 2 3
```

Out[3]:

```
┌──────┐
│1 2 3│
└~────→┘
```

A dfn using "conventional" logic as as follows:

In [4]:

```
]disp {1<⍳a←⍴⍴⍵: ⊂[⍳a]⍵ ⋄ ⊂⍵} 1 2 3
```

Out[4]:

```
┌──────┐
│1 2 3│
└~────→┘
```

And finally, using split:

In [5]:

```
]disp ↓1 2 3
```

Out[5]:

```
┌──────┐
│1 2 3│
└~────→┘
```

Tasks 8 #11 #12 #13

Convert integers to booleans in several ways:

In [2]:

```
{b←(⌈/⍵)⍴0 ⋄ b[⍵]←1 ⋄ b} 1 3 7 3 7 8
```

Out[2]:

```
1 0 1 0 0 0 1 1
```

In [3]:

```
{b←(⌈/⍵)⍴0 ⋄ ((⊂⍵)⌷b)←1 ⋄ b} 1 3 7 3 7 8
```

Out[3]:

```
1 0 1 0 0 0 1 1
```

In [6]:

```
MaskAt ← {⍵∊⍨⍳⌈/⍵}
MaskAt 1 3 7 3 7 8
```

Out[6]:

```
1 0 1 0 0 0 1 1
```

The `MaskAt` function can be used with *partitioned-enclose* to achieve the `SplitAt` function:

In [7]:

```
1 3 3 7 {((⍴⍵)↑MaskAt ⍺)⊂⍵} 12↑⎕A
```

Out[7]:

```
┌──┬────┬──────┐
│AB│CDEF│GHIJKL│
```

```
╵  ╵     ╵      ╵
```

However, we can simply use the *ideas* in `MaskAt` directly in our `SplitAt` function:

In [8]:

```
1 3 3 7 {((⍳⍴⍵)∊⍺)⊂⍵} 9↑⎕A
```

Out[8]:

```
┌──┬────┬───┐
│AB│CDEF│GHI│
└──┴────┴───┘
```

Remove parentheses using the commute (swap) operator ⍨

In [9]:

```
1 3 3 7 {⍵⊂⍨⍺∊⍨⍳⍴⍵} 9↑⎕A
```

Out[9]:

```
┌──┬────┬───┐
│AB│CDEF│GHI│
└──┴────┴───┘
```

An alternative coding uses an outer product for comparison.

In [13]:

```
3 7 {⍵⊂⍨+⌿⍺∘.=⍳⍴⍵} 9↑⎕A
```

Out[13]:

```
┌────┬───┐
│CDEF│GHI│
└────┴───┘
```

But watch out for the edge cases of a scalar left argument, or duplicate elements in the left argument!

In [15]:

```
7 {⍵⊂⍨+⌿⍺∘.=⍳⍴⍵} 9↑⎕A
```

Out[15]:

```
┌─┬─┬─┬─┬─┬─┬─┬─┬─┐
│A│B│C│D│E│F│G│H│I│
└─┴─┴─┴─┴─┴─┴─┴─┴─┘
```

In [16]:

```
1 3 3 7 {⍵⊂⍨+⌿⍺∘.=⍳⍴⍵} 9↑⎕A
```

Out[16]:

```
┌──┬┬────┬───┐
│AB││CDEF│GHI│
└──┴┴────┴───┘
```

Fix with **or-reduce** and **ravel**:

In [17]:

```
7 {⍵⊂⍨∨⌿(,⍺)∘.=⍳⍴⍵} 9↑⎕A
```

Out[17]:

```
┌───┐
│GHI│
└───┘
```

```
In [18]:  1 3 3 7 {ω⊂⍨∨⌿(,α)∘.=⍳⍴ω} 9↑⎕A

Out[18]:  ┌──┬────┬───┐
          │AB│CDEF│GHI│
          └──┴────┴───┘
```

Split on delimiter

This is most easily achieved with the "partition" function. That is represented by dyadic `⊆` in Dyalog Unicode version 16.0 onwards. In Classic, we can use `Part←{α ⎕U2286 ω}`. For versions prior to 16.0, use `Part←{⎕ML←3 ⋄ α⊂ω}`.

```
In [11]:  Part←{⎕ML←3 ⋄ α⊂ω}
          ' ' {(α≠ω)Partω} 'hello world'

Out[11]:  ┌─────┬─────┐
          │hello│world│
          └─────┴─────┘
```

For multiple delimiters, use not-membership:

```
In [12]:  '#!' {(~ω∊α)Partω} 'well#hello!there'

Out[12]:  ┌────┬─────┬─────┐
          │well│hello│there│
          └────┴─────┴─────┘
```

Tasks8 #17

One solution converts the argument into a list of numbers and references to the root namespace (#).

```
In [13]:  {((⍎ω)≠⍎'#')/⍎ω} '14#32#612#4#54'

Out[13]:  14 32 612 4 54
```

However, this solution does not generalise to other delimiters. In this case, we would likely want to modify a temporary array.

```
In [14]:  {r←ω ⋄ (('#'=r)/r)←' ' ⋄ ,⍎r}'42#31#216'

Out[14]:  42 31 216
```

```
In [15]:  {r←ω ⋄ r[m/⍳⍴m←r='#']←' ' ⋄ ,⍎r}'42#31#216'

Out[15]:  42 31 216
```

This same technique will still work, even if we use `'%'` or any other character instead of `'#'` .

In [16]:

```
'%' {r←⍵ ⋄ ((⍺=r)/r)←' ' ⋄ ,⍎r} '42%31%216'
```

Out[16]:

```
42 31 216
```

In [17]:

```
'$' {r←⍵ ⋄ ((⍺=r)/r)←' ' ⋄ ,⍎r} '42$31$216'
```

Out[17]:

```
42 31 216
```

Tasks8 #18

We need to split on delimiter(s) and execute each. We use ravel `,⍵` to ensure our arguments and results are vectors.

In [20]:

```
]display {,¨⍎¨(⍵≠',')Part ,⍵} '1 2,5 3 2,4'
```

Out[20]:

```
┌→──────────────────┐
│ ┌→──┐ ┌→────┐ ┌→┐ │
│ │1 2│ │5 3 2│ │4│ │
│ └~──┘ └~────┘ └~┘ │
└∊──────────────────┘
```

In [22]:

```
]display {,¨⍎¨ (~⍵∊',#')Part ,⍵} '1 2,5 3 2#4'
```

Out[22]:

```
┌→──────────────────┐
│ ┌→──┐ ┌→────┐ ┌→┐ │
│ │1 2│ │5 3 2│ │4│ │
│ └~──┘ └~────┘ └~┘ │
└∊──────────────────┘
```

Tasks 8 #19

In [23]:

```
(2 1 5⍴1 3 2 5 3) {↑(↓⍺)∘.×(↓⍵)} (2 5⍴2 3 5 2)
```

Out[23]:

```
2  9 10 10 6
3 15  4 10 9



2  9 10 10 6
3 15  4 10 9
```

In recent versions of Dyalog, we can use the rank operator (http://help.dyalog.com/latest/#Language/Primitive%20Operators/Rank.htm).

In [24]:

```
(2 1 5⍴1 3 2 5 3) (×⍤1⍤1 99) (2 5⍴2 3 5 2)
```

Out[24]:

```
2  9 10 10 6
3 15  4 10 9
```

```
2  9 10 10 6
3 15  4 10 9
```