

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 $\alpha \leftarrow \omega$ and partition $\alpha \subseteq \omega$

Migration Level

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

Prior to Dyalog version 16.0, use $\text{Partition} \leftarrow \{\bar{\square} \text{ML} \leftarrow 3 \diamond \alpha \leftarrow \omega\}$

From version 16.0 onwards, use $\text{Partition} \leftarrow \{\alpha \bar{\square} \text{U2286 } \omega\}$

The monadic function **enclose-if-simple** can be encoded as a dfn: $\text{EIS} \leftarrow \{1 \geq \omega : \leftarrow \omega \diamond \omega\}$

From version 16.0 onwards, use $\text{EIS} \leftarrow \{\bar{\square} \text{U2286 } \omega\}$

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 \downarrow). 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←pparray
  :If rank>1
    array←c[rank]array
  :Else
    array←carray
  :EndIf
▽
```

In [8]:

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

Out[8]:

```
┌───┐
```

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \end{array}$$

```
]disp {c[(0<r)/r<=rho*omega]} 1 2 3
```

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \end{array}$$

A dfn using "conventional" logic as as follows:

```
]disp {1<la←ppw: c[la]w ◇ cω} 1 2 3
```

1	2	3
---	---	---

And finally, using split:

```
]disp ↓ 1 2 3
```

1	2	3
---	---	---

Tasks 8 #11 #12 #13

Convert integers to booleans in several ways:

$$\{b \leftarrow (\lceil \cdot / \omega \rceil) p_0 \diamond b[\omega] \leftarrow 1 \diamond b\} \quad 1 \quad 3 \quad 7 \quad 3 \quad 7 \quad 8$$

1 0 1 0 0 0 1 1

$$\{b \leftarrow (\lceil / \omega) \rho_0 \diamond ((\leq \omega) \square b) \leftarrow 1 \diamond b\} \quad 1 \quad 3 \quad 7 \quad 3 \quad 7 \quad 8$$

1 0 1 0 0 0 1 1

$$\text{MaskAt} \leftarrow \{\omega \in \mathbb{Z}^+ \mid \omega \mid \omega\}$$

$$\text{MaskAt} \quad 1 \quad 3 \quad 7 \quad 3 \quad 7 \quad 8$$

1 0 1 0 0 0 1 1

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

$$1 \ 3 \ 3 \ 7 \ \{((p\omega)\uparrow \text{MaskAt } \alpha) \in \omega\} \ 12 \uparrow \Box A$$

AB	CDEF	GHIJKL
----	------	--------

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

In [8]: `1 3 3 7 {((ιρω)εα)εω} 9↑□A`

Out[8]:

AB	CDEF	GHI

Remove parentheses using the commute (swap) operator $\tilde{\sim}$

In [9]: `1 3 3 7 {ωε̃αε̃ιρω} 9↑□A`

Out[9]:

AB	CDEF	GHI

An alternative coding uses an outer product for comparison.

In [13]: `3 7 {ωε̃+α°.=ιρω} 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 {ωε̃+α°.=ιρω} 9↑□A`

Out[15]:

A	B	C	D	E	F	G	H	I

In [16]: `1 3 3 7 {ωε̃+α°.=ιρω} 9↑□A`

Out[16]:

AB		CDEF	GHI

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

In [17]: `7 {ωε̃∨(,α)°.=ιρω} 9↑□A`

Out[17]:

GHI

In [18]: `1 3 3 7 {ω<~v f(,α)∘.=ιρω} 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/ιρm←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←ω ⋄ ((α=r)/r)←' ' ⋄ ,⊥r} '42%31%216'
```

```
Out[16]: 42 31 216
```

```
In [17]: '$' {r←ω ⋄ ((α=r)/r)←' ' ⋄ ,⊥r} '42$31$216'
```

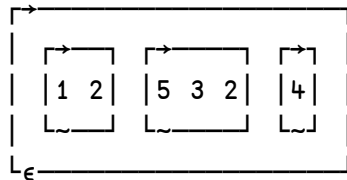
```
Out[17]: 42 31 216
```

Tasks8 #18

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

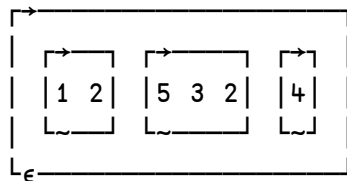
```
In [20]: ]display {,¨⊥¨(ω≠',')Part ,ω} '1 2,5 3 2,4'
```

```
Out[20]:
```



```
In [22]: ]display {,¨⊥¨ (~ω≠',#')Part ,ω} '1 2,5 3 2#4'
```

```
Out[22]:
```



Tasks 8 #19

```
In [23]: (2 1 5ρ1 3 2 5 3) {↑(↓α)∘.×(↓ω)} (2 5ρ2 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) (<http://help.dyalog.com/latest/#Language/Primitive%20Operators/Rank.htm>).

```
In [24]: (2 1 5ρ1 3 2 5 3) (×%1%1 99) (2 5ρ2 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