10 LAMBDA functions to partition an array into 2 thunks

A cautionary tale

Owen Price

I had an idea to create a LAMBDA function that would simply and easily split an array into two parts:

1) the first n rows, and
2) the rest of the rows

I figured it would return a two-row array, with each output cell containing a thunk holding either 1) or 2) from above.

But as is sometimes the case, things quickly got out of hand.

Microsoft® MVP Most Valuable Professional
Owen Price

Here's what it does

Input array

1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30	31	32	33
34	35	36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63	64	65	66
67	68	69	70	71	72	73	74	75	76	77
78	79	80	81	82	83	84	85	86	87	88
89	90	91	92	93	94	95	96	97	98	99
100	101	102	103	104	105	106	107	108	109	110

PARTITION > B15

```
=LET(
    parts, PARTITION_1(B3#, 3),
    part1, GETPARTITION(parts, 1),
    part2, GETPARTITION(parts, 2),
    IFERROR(HSTACK(part1, part2), "")
)
```

Partition the array, and restructure it

1	2	3	4	5	6	7	8	9	10	11	34	35	36	37	38	39	40	41	42	43	44
12	13	14	15	16	17	18	19	20	21	22	45	46	47	48	49	50	51	52	53	54	55
23	24	25	26	27	28	29	30	31	32	33	56	57	58	59	60	61	62	63	64	65	66
											67	68	69	70	71	72	73	74	75	76	77
											78	79	80	81	82	83	84	85	86	87	88
											89	90	91	92	93	94	95	96	97	98	99
											100	101	102	103	104	105	106	107	108	109	110

Microsoft® Most Valuable Professional

Owen Price

But that’s somewhat beside the point

Owen Price

I started with this simple function for partitioning on rows

```
PARTITION = LAMBDA(array, n,
    LET(
        part1, TAKE(array, n),
        part2, DROP(array, n),
        VSTACK(LAMBDA(part1), LAMBDA(part2))
    )
);
```

Wrapping an expression in LAMBDA without any parameters is “thunking” that expression. It stores the expression without evaluating it

Microsoft® MVP Most Valuable Professional

Owen Price

Then I added an axis parameter, to let the user choose to partition on columns instead

```
PARTITION = LAMBDA(array, n, [axis],
    LET(
        _axis, IF(ISOMITTED(axis), 0, 1),
        part1, IF(_axis=0, TAKE(array, n), TAKE(array,,n)),
        part2, IF(_axis=0, DROP(array, n), DROP(array,,n)),
        VSTACK(LAMBDA(part1), LAMBDA(part2))
    )
);
```

TAKE and DROP both have ‘rows’ and ‘columns’ for their second and third parameters

And of course partitioning on columns should output a horizontal array

```
PARTITION = LAMBDA(array, n, [axis],
    LET(
        _axis, IF(ISOMITTED(axis), 0, 1),
        part1, IF(_axis=0, TAKE(array, n), TAKE(array,,n)),
        part2, IF(_axis=0, DROP(array, n), DROP(array,,n)),
        IF(
            _axis=0,
            VSTACK(LAMBDA(part1), LAMBDA(part2)),
            HSTACK(LAMBDA(part1), LAMBDA(part2))
        )
    )
);
```

Owen Price

But all that duplication is no good. Let's create a stack_fn variable that selects the stacking function

```
PARTITION = LAMBDA(array, n, [axis],
    LET(
        _axis, IF(ISOMITTED(axis), 0, 1),
        part1, IF(_axis=0, TAKE(array, n), TAKE(array,,n)),
        part2, IF(_axis=0, DROP(array, n), DROP(array,,n)),
        stack_fn, IF(_axis=0, VSTACK, HSTACK),
        stack_fn(LAMBDA(part1), LAMBDA(part2))
    )
);
```

Assigning the result of the stack_fn IF to a variable means the variable is now either VSTACK or HSTACK

Owen Price

But we don’t actually need to name it. Let’s just use the IF formula where we would otherwise put VSTACK or HSTACK

```
PARTITION = LAMBDA(array, n, [axis],
    LET(
        _axis, IF(ISOMITTED(axis), 0, 1),
        part1, IF(_axis=0, TAKE(array, n), TAKE(array,,n)),
        part2, IF(_axis=0, DROP(array, n), DROP(array,,n)),
        IF(_axis=0, VSTACK, HSTACK)(LAMBDA(part1), LAMBDA(part2))
    )
);
```

Anywhere we can use a function, we can also use an expression that returns a function

And since omitted optional arguments default to zero, we can get rid of the _axis variable

```
PARTITION = LAMBDA(array, n, [axis],
    LET(
        part1, IF(axis=0, TAKE(array, n), TAKE(array,,n)),
        part2, IF(axis=0, DROP(array, n), DROP(array,,n)),
        IF(axis=0, VSTACK, HSTACK)(LAMBDA(part1), LAMBDA(part2))
    )
);
```


Owen Price

To keep things clean, let's move the thunking and embed it in the part definitions

```
PARTITION = LAMBDA(array, n, [axis],
    LET(
        part1, LAMBDA(IF(axis=0, TAKE(array, n), TAKE(array,,n))),
        part2, LAMBDA(IF(axis=0, DROP(array, n), DROP(array,,n))),
        IF(axis=0, VSTACK, HSTACK)(part1, part2)
    )
);
```

Microsoft® Most Valuable Professional

Owen Price

But the parts only vary on TAKE or DROP, so let's create a part_fn function that accepts a function and returns a function

```
PARTITION = LAMBDA(array, n, [axis],
    LET(
        part_fn, LAMBDA(f,
            LAMBDA(IF(axis=0, f(array, n), f(array,,n)))
        ),
        part1, part_fn(TAKE),
        part2, part_fn(DROP),
        IF(axis=0, VSTACK, HSTACK)(part1, part2)
    )
);
```

Since TAKE and DROP both have 'rows' and 'columns' for their second and third parameters, we can call them in the same way with this parameter "f"

And let's forget about the part variables and just embed the calls to part_fn in the stacking operation

```
PARTITION = LAMBDA(array, n, [axis],
    LET(
        part_fn, LAMBDA(f,
            LAMBDA(IF(axis=0, f(array, n), f(array,,n)))
        ),
        IF(axis=0, VSTACK, HSTACK)(part_fn(TAKE), part_fn(DROP))
    )
);
```

Microsoft® MVP Most Valuable Professional
Owen Price

Or even better, just map the part function over an array of TAKE and DROP, whose orientation is determined by the axis argument.

```
PARTITION = LAMBDA(array, n, [axis],
    MAP(
        IF(axis=0,VSTACK,HSTACK)(TAKE,DROP),
        LAMBDA(f, LAMBDA(IF(axis=0,f(array,n),f(array,,n))))
    )
);
```

The IF expression returns either VSTACK(TAKE,DROP) or HSTACK(TAKE,DROP) depending on axis. Each of those are then mapped to the parameter f, where they are passed to the inner function and called

Microsoft® MVP Most Valuable Professional

Owen Price

Finally, as is tradition, let's curry the axis argument so we can create some derived functions

```
PARTITION = LAMBDA([axis],
    LAMBDA(array, n,
        MAP(
            IF(axis=0,VSTACK,HSTACK)(TAKE,DROP),
            LAMBDA(f, LAMBDA(IF(axis=0,f(array,n),f(array,,n))))
        )
    )
);

PARTITIONROWS = PARTITION(0);
PARTITIONCOLS = PARTITION(1);
```

Each of PARTITIONROWS and PARTITIONCOLS returns the LAMBDA(array, n function from the PARTITION definition

Microsoft® Most Valuable Professional

Owen Price

I enjoy these kinds of exercises.

They’re great for learning!

Occasionally they become practically useful.

Though the final version of PARTITION is more concise and has more functionality than the earlier versions, is it easier to understand?

I suppose the moral of the story is →

Just because you can, doesn’t mean you should!

Owen Price