

DYALOG

APL Array Notation

Adám Brudzewsky

*Head of Language Design
Dyalog Ltd.*



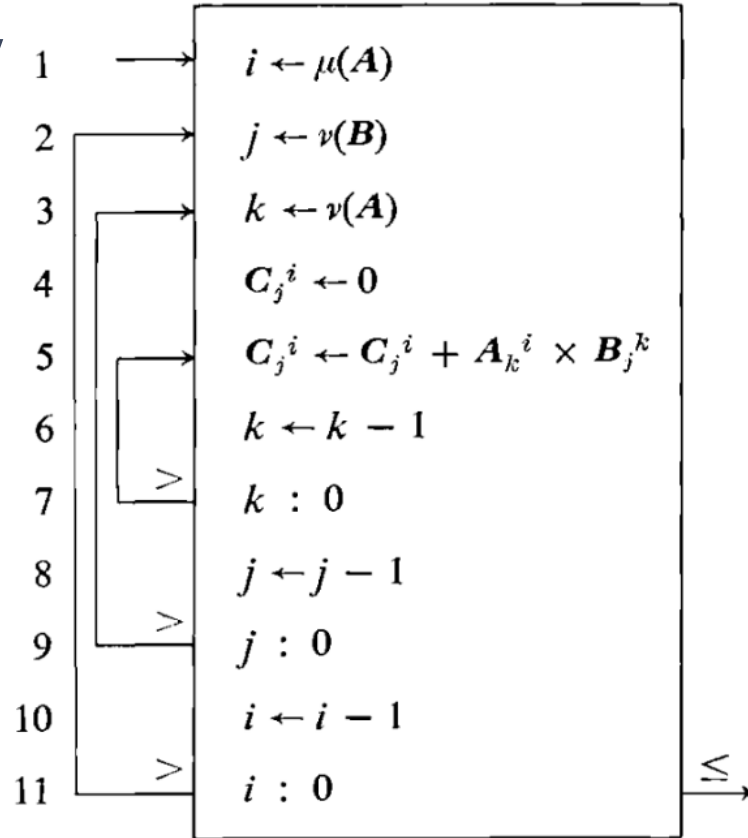
A Plan?

- History
- APL
- Problem
- Design Iteration
- Specification
- Implementation
- Standardisation?
- Take-aways?

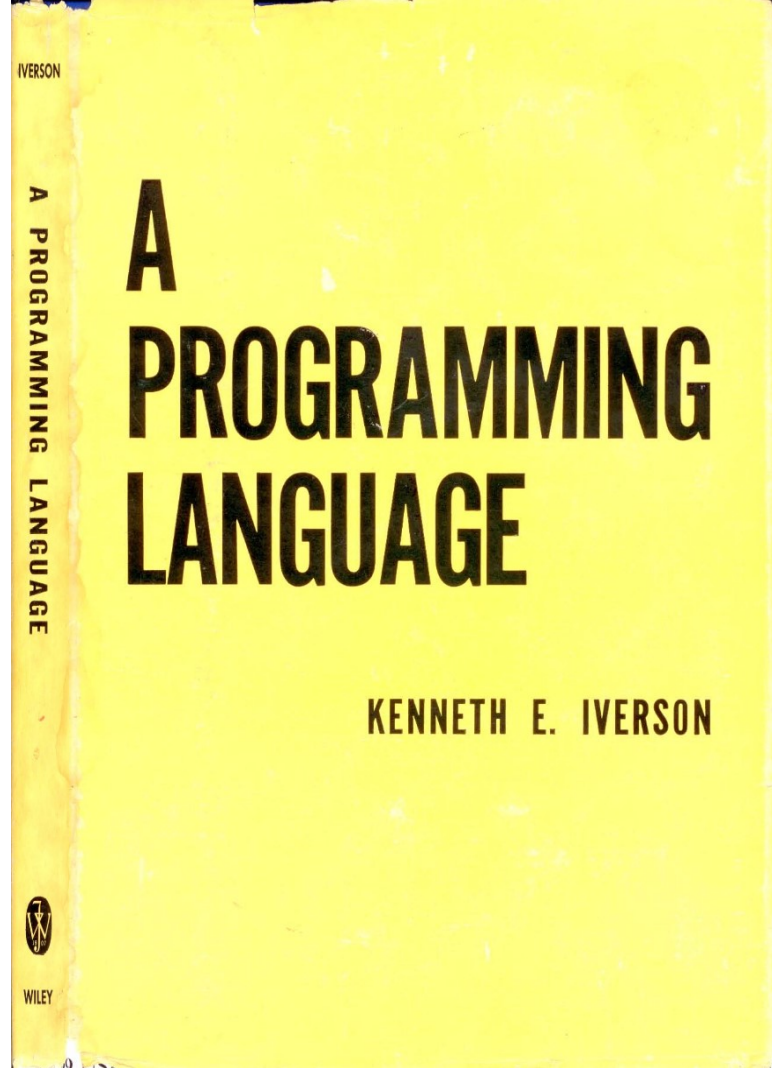


History

-1962:



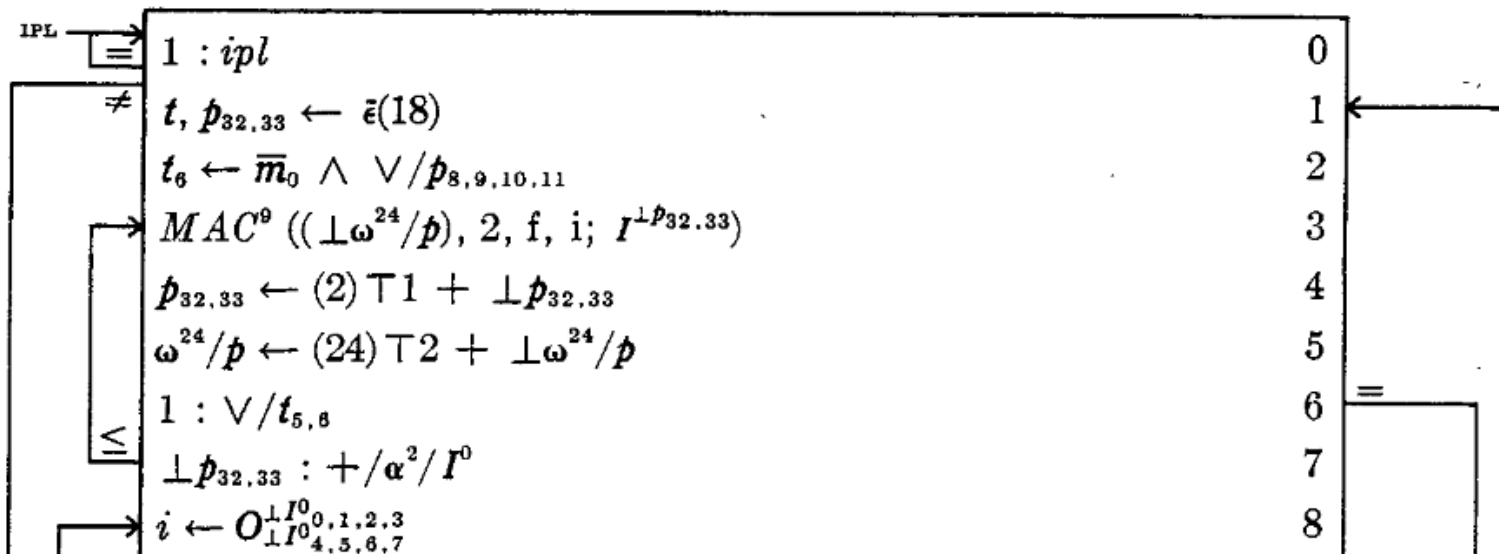
Program 1.5 Matrix multiplication



History

-1964: IBM System/360 design

CPU, central processing unit system program



-1966: IBM in-house

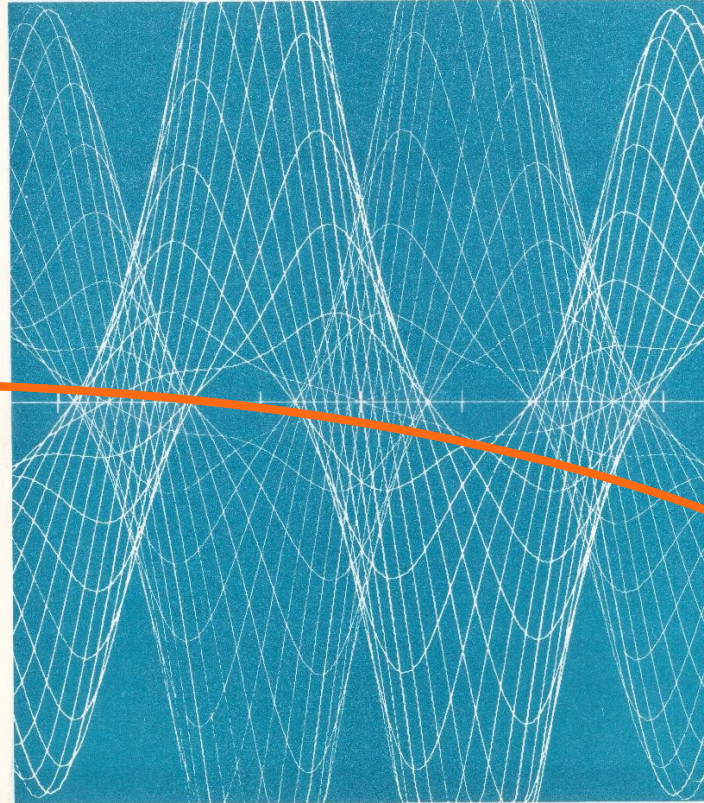
```

3333330000000000\\\\\\LLLLLPPPPP66666\\\\\\AAAAAAAAAAAAAAAAAAAA 66666
3333330000000000\\\\\\LLLLLPPPPP66666\\\\\\AAAAAAAAAAAAAAAAAAAA 66666
3333330000000000\\\\\\LLLLLPPPPP66666\\\\\\AAAAAAAAAAAAAAAAAAAA 66666
PPPPP          AAAAA66666\\\\\\66666\\\\\\00000 66666LLLLL
PPPPP          AAAAA66666\\\\\\66666\\\\\\00000 66666LLLLL
PPPPP          AAAAA66666\\\\\\66666\\\\\\00000 66666LLLLL
      LLLLL66666 00000 AAAAA\\\\\\\\\\\\\\ 00000 \\\\\\\\
      LLLLL66666 00000 AAAAA\\\\\\\\\\\\\\ 00000 \\\\\\\\
      LLLLL66666 00000 AAAAA\\\\\\\\\\\\\\ 00000 \\\\\\\\
AAAAA0000066666 66666 33333 6666600000 AAAAA
AAAAA0000066666 66666 33333 6666600000 AAAAA
AAAAA0000066666 66666 33333 6666600000 AAAAA
      PPPPP66666LLLLL00000\\\\\\\\ PPPPP\\\\\\\\ AAAAA00000
      PPPPP66666LLLLL00000\\\\\\\\ PPPPP\\\\\\\\ AAAAA00000
      PPPPP66666LLLLL00000\\\\\\\\ PPPPP\\\\\\\\ AAAAA00000
LLLLL33333PPPPAAAAA\\\\\\\\ PPPPP 6666600000\\\\\\\\
LLLLL33333PPPPAAAAA\\\\\\\\ PPPPP 6666600000\\\\\\\\
LLLLL33333PPPPAAAAA\\\\\\\\ PPPPP 6666600000\\\\\\\\
33333      LLLLLLLLLL\\\\\\\\AAAAALLLLL AAAAA\\\\\\\\\\\\\\\\
33333      LLLLLLLLLL\\\\\\\\AAAAALLLLL AAAAA\\\\\\\\\\\\\\\\
33333      LLLLLLLLLL\\\\\\\\AAAAALLLLL AAAAA\\\\\\\\\\\\\\\\
      AAAAA\\\\\\\\333366666AAAAALLLLL33333PPPPP 66666 00000
      AAAAA\\\\\\\\333366666AAAAALLLLL33333PPPPP 66666 00000

```

History

-1968: Father



RC 1922

THE APL 360 TERMINAL SYSTEM

A. D. Falkoff/ K. E. Iverson

October 16, 1967
Third Printing, March, 1968

IBM RESEARCH

History

-1972: Father teaches and consults

A 3497/77

Anm. 29. aug. 1977 kl. 13

MEANS EASIER COMMUNICATION MEANS FASTER
CODING MEANS FEWER CODERS

Henri Brudzewsky, rådgivende ingeniørvirksomhed, Mindevej 28, Søborg,

klasse 9, herunder datamaskiner, især APL-kodede datamaskiner,

klasse 42: EDB-servicebureauvirksomhed, især under anvendelse af APL-kodede datamaskiner.

History

-1986: My first APL conference



History

-2014: I join Dyalog



History

youtu.be/9-HAvTMhYao



```
(...)  
'second row of matrix'  
'third row of matrix']  
(...)
```

A Doesn't have to be brackets – many reasons why not

A [♦] (♦) [[♦]] ([♦]) ⌈ ♦ ⌋

A Problem with digraphs – human parsing – unicode – many bracketing pairs.

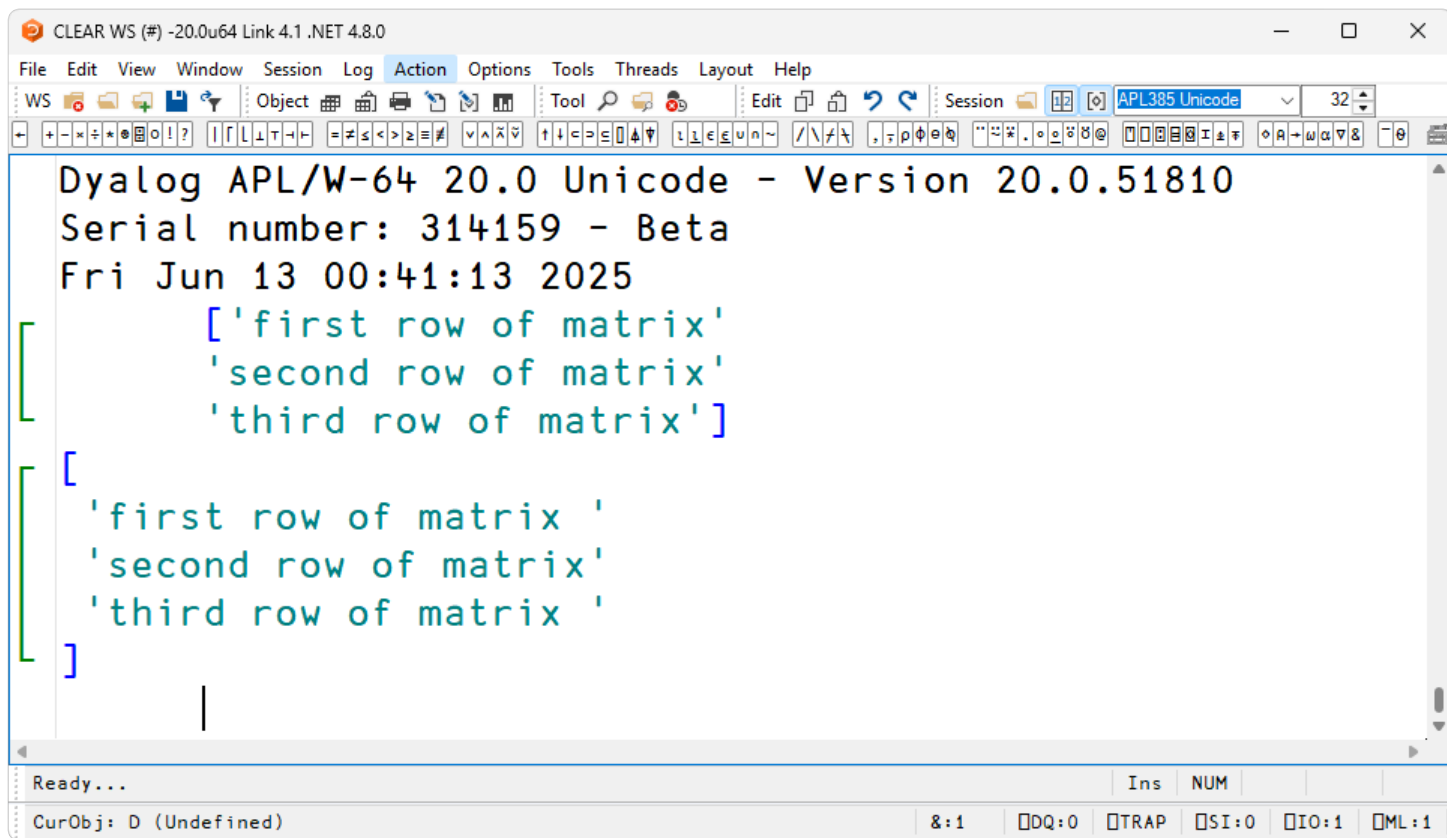
```
(...)
```

-2015: My hope is that Dyalog and as many as possible of other APL vendors will consider these proposals and questions and perhaps come to some kind of agreement to include at least the basic concepts into their products.

Phil Last, <phil.last@4xtra.com> Wiltshire, Wednesday 9 Sept 2015

History

-2025: PLSS



The screenshot shows the CLEAR WS APL editor window. The title bar reads "CLEAR WS (#) -20.0u64 Link 4.1 .NET 4.8.0". The menu bar includes File, Edit, View, Window, Session, Log, Action, Options, Tools, Threads, Layout, and Help. The toolbar contains various icons for file operations, editing, and session management. The main text area displays the following APL code:

```
Dyalog APL/W-64 20.0 Unicode - Version 20.0.51810
Serial number: 314159 - Beta
Fri Jun 13 00:41:13 2025

[
  ['first row of matrix'
   'second row of matrix'
   'third row of matrix']
[
  ['first row of matrix '
   'second row of matrix'
   'third row of matrix '
  ]
]
```

The status bar at the bottom shows "Ready..." and "CurObj: D (Undefined)". On the right side of the status bar, there are several indicators: "Ins", "NUM", "8:1", "DQ:0", "TRAP", "SI:0", "IO:1", and "ML:1".

History

- 1962: Kenneth E. Iverson, *A Programming Language*
- 1964: IBM *System/360* design
- 1966: IBM in-house APL\360
- 1968: Father joins IBM Research
- 1972: Father teaches and consults
- 1986: My first APL conference
- 2014: I join Dyalog
- 2015: Phil Last, *APL Array Notation*
- 2025: PLSS

APL



```
1 2 3 4 5 6
2 3
2 3 + 40
42 43
2 3 + 40 50
42 53
2 3 + (40 50) (60 70)
42 52 63 73
```

CC BY-SA 4.0: w.wiki/EY\$

APL

```
1 2 3
4 5 6
```

2 3 ρ ι 6

APL

2 3 ρ ι 6 ♦ 2 3 ρ 6 + ι 6

1	2	3
4	5	6
7	8	9
10	11	12

APL

$(2 \ 3 \ \rho \ 1 \ 6) \ (2 \ 3 \ \rho \ 6 \ + \ 1 \ 6)$

1	2	3	7	8	9
4	5	6	10	11	12

APL

```
      (2 3 ρ ι 6) (2 3 ρ 6 + ι 6)  
1 2 3      7      8      9  
4 5 6     10     11     12
```

```
      X  
( 1 2 3 ( 7 8 9  
 4 5 6 ) 10 11 12)
```

© 1981 STSC, Inc.

The variable *X* contains a two-item vector, each of whose items is a two-row, three-column matrix

APL

$$\begin{array}{cccccc} & & (2 & 3 & \rho & \iota & 6) & (2 & 3 & \rho & 6 & + & \iota & 6) \\ 1 & 2 & 3 & & 7 & 8 & 9 & & & & & & & \\ 4 & 5 & 6 & & 10 & 11 & 12 & & & & & & & \end{array}$$

$$\begin{array}{cccc} & & X & \\ (& 1 & 2 & 3 & (& 7 & 8 & 9 \\ & & & & 4 & 5 & 6) & 10 & 11 & 12) \end{array}$$

© 1981 STSC, Inc.

$$1 \ 2 \ 3 \ (2 \ 3 \ \rho \ 7 \ 8 \ 9 \ 4 \ 5 \ 6) \ 10 \ 11 \ 12$$

APL

```

      (2 3 ρ ι 6) (2 3 ρ 6 + ι 6)
1 2 3      7 8 9
4 5 6    10 11 12

```

```

      X
( 1 2 3 ( 7 8 9
      4 5 6 ) 10 11 12 ) © 1981 STSC, Inc.

```

```

      1 2 3 (2 3 ρ 7 8 9 4 5 6) 10 11 12
1 2 3 7 8 9 10 11 12
      4 5 6

```

Problem

$(2 \ 3 \ \rho \ \iota \ 6) \ (2 \ 3 \ \rho \ 6 \ + \ \iota \ 6)$

1	2	3	7	8	9
4	5	6	10	11	12

Problem

$(2\ 3\ \rho\ \iota\ 6)\ (2\ 3\ \rho\ 6\ +\ \iota\ 6)$

1	2	3	7	8	9
4	5	6	10	11	12

Problem

$(2\ 3\ \rho\ \iota\ 6)\ (2\ 3\ \rho\ 6\ +\ \iota\ 6)$

1	2	3	7	8	9
4	5	6	10	11	12

Problem

- ◆ Intuitive
- ◆ Easy to type
- ◆ Ideally ASCII
- ◆ Versatile (single/multi-line, Table, nested, object)
- ◆ Unambiguous — consider:
`list[i]   table[i;j]   (1 2)(3 4)   {lambda}`

Design Iteration

2015

Acre, Mk I

Table: [1 2 3 ♦ 4 5 6]

Design Iteration

-2015

Acre, Mk I

Table:

```
[ 1  2  3
  4  5  6]
```

Design Iteration

2015

Acre, Mk I

Table:

1	2	3
4	5	6

Design Iteration

2015

Acre, Mk I

Table: [1 2 3 ♦ 4 5 6]

Design Iteration

2015

Acre, Mk I

Table: [1 2 3 ♦ 4 5 6]

List: —

Design Iteration

2015

Acre, Mk I

Table: [1 2 3 ♦ 4 5 6]

List: —

Object: [[name1←val1 ♦ name2←val2]]

Design Iteration

-2015

Acre, Mk I

Table: [1 2 3 ♦ 4 5 6]

List: —

Object: [[name1←val1 ♦ name2←val2]]

Value: 6×a←7

Design Iteration

2015

Acre, Mk I

Table: [1 2 3 ♦ 4 5 6]

List: —

Object: [[name1←val1 ♦ name2←val2]]

Value: 6×a←7

Table: [a←7 ♦ b←8]

Design Iteration

-2015

Acre, Mk I

Table: [1 2 3 ♦ 4 5 6]

List: —

Object: [[name1←val1 ♦ name2←val2]]

Value: 6×a←7

Table: [a←7 ♦ b←8]

Indexing: list[indices]

Design Iteration

-2015

Acre, Mk I

Table: [1 2 3 ♦ 4 5 6]

List: —

Object: [[name1←val1 ♦ name2←val2]]

Value: 6×a←7

Table: [a←7 ♦ b←8]

Indexing: list[indices]

Ambiguous: list [[a←7 ♦ b←8]]

Design Iteration

-2015

Acre, Mk I

Table: [1 2 3 ♦ 4 5 6]

List: —

Object: [[name1←val1 ♦ name2←val2]]

Value: 6×a←7

Table: [a←7 ♦ b←8]

Indexing: list[indices]

Ambiguous: list([[a←7 ♦ b←8]])

Design Iteration

		<i>Acre, Mk II</i>
-2018	Table:	[1 2 3 ♦ 4 5 6]
	List:	—
	Object:	[name1←val1 ♦ name2←val2]

Design Iteration

		Acre, Mk II
-2018	Table:	[1 2 3 ♦ 4 5 6]
	List:	—
	Object:	[name1←val1 ♦ name2←val2]
	Empty obj?	[]

Design Iteration

		<i>Acre, Mk II</i>
-2018	Table:	[1 2 3 ♦ 4 5 6]
	List:	—
	Object:	[name1←val1 ♦ name2←val2]
	Empty obj?	[]
	All rows:	table[; 4 2]

Design Iteration

		<i>Acre, Mk II</i>
-2018	Table:	[1 2 3 ♦ 4 5 6]
	List:	—
	Object:	[name1←val1 ♦ name2←val2]
	Empty obj?	[]
	All rows:	table[; 4 2]
	Ambiguous:	list []

Design Iteration

		<i>Acre, Mk II</i>
-2018	Table:	[1 2 3 ♦ 4 5 6]
	List:	—
	Object:	[name1←val1 ♦ name2←val2]
	Empty obj?	[]
	All rows:	table[; 4 2]
	Ambiguous:	list([])

Design Iteration

		<i>Acre, Mk II</i>
-2018	Table:	[1 2 3 ♦ 4 5 6]
	List:	—
	Object:	[name1←val1 ♦ name2←val2]
	Empty obj?	[]
	All rows:	table[; 4 2]
	Ambiguous: list	[]
	Empty obj:	[← ♦]

Design Iteration

		Dyalog, <i>Mk I</i>
-2017	Table:	(1 2 3 ♦ 4 5 6)
	List:	—
	Object:	(name1:val1 ♦ name2:val2)

Design Iteration

		Dyalog, <i>Mk I</i>
-2017	Table:	(1 2 3 ♦ 4 5 6)
	List:	—
	Object:	(name1:val1 ♦ name2:val2)
	List:	(1 ♦ 4 ♦ 7)

Design Iteration

Dyalog, *Mk I*

-2017

Table: (1 2 3 ♦ 4 5 6)
List: —
Object: (name1:val1 ♦ name2:val2)
List: (1 ♦ 4 ♦ 7)
Column: ((1 ♦) ♦ (4 ♦) ♦ (7 ♦))

Design Iteration

Dyalog, *Mk I*

-2017

Table: (1 2 3 ♦ 4 5 6)
List: —
Object: (name1:val1 ♦ name2:val2)
List: (1 ♦ 4 ♦ 7)
Column: ((1 ♦) ♦ (4 ♦) ♦ (7 ♦))
Column? (1 ♦ 4 ♦ 7)

Design Iteration

Dyalog, *Mk I*

-2017

Table: (1 2 3 ♦ 4 5 6)
List: —
Object: (name1:val1 ♦ name2:val2)
List: (1 ♦ 4 ♦ 7)
Column: ((1 ♦) ♦ (4 ♦) ♦ (7 ♦))
Column? (1 ♦ 4 ♦ 7)
List? —

Design Iteration

Dyalog, *Mk III*

Table: [1 2 3 ♦ 4 5 6]

List: (1 2 3 ♦ 1 2)

Object: (name1:val1 ♦ name2:val2)

2020

Design Iteration

Dyalog, *Mk III*

-2020

Table: [1 2 3 ♦ 4 5 6]

List: (1 2 3 ♦ 1 2)

Object: (name1:val1 ♦ name2:val2)

Column: [1 ♦ 4 ♦ 7]

Design Iteration

Dyalog, *Mk III*

2020

Table: [1 2 3 ♦ 4 5 6]

List: (1 2 3 ♦ 1 2)

Object: (name1 : val 1 ♦ name2 : val 2)

Column: [1
4
7]

Design Iteration

Dyalog, *Mk III*

-2020

Table: [1 2 3 ♦ 4 5 6]

List: (1 2 3 ♦ 1 2)

Object: (name1:val1 ♦ name2:val2)

Column: [1
4
7]

Empty obj: ()

Design Iteration

- 2015: Phil Last, *APL Array Notation* model
- 2016: Acre-3, *Mk I* APL model; Dyalog, *Mk I* internal
- 2017: Dyalog, *Mk II* presentation
- 2018: Acre-4, *Mk II* APL model; Dyalog, *Mk III* presentation
- 2020: Dyalog, *RC1* presentation
- 2021: Dyalog Formal Specification & Community Feedback
- 2022: Dyalog, *RC1* APL model
- 2023: Dyalog, *RC1* C prototype
- 2025: Dyalog, *RC2* implementation

Design Iteration

- 2015: Phil Last, *APL Array Notation* model
- 2016: Acre-3, *Mk I* APL model; Dyalog, *Mk I* internal
- 2017: Dyalog, *Mk II* presentation
- 2018: Acre-4, *Mk II* APL model; Dyalog, *Mk III* presentation
- 2020: Dyalog, *RC1* presentation
- 2021: Dyalog Formal Specification & Community Feedback**
- 2022: Dyalog, *RC1* APL model
- 2023: Dyalog, *RC1* C Prototype
- 2025: Dyalog, *RC2* Implementation

Formal Proposal: APL Array Notation

Note: This proposal's specification for [scoping in literal namespaces](#) has been changed based on feedback received after its initial publication. Affected sections are marked by a vertical bar on the left, with removed text ~~struck through~~ and added text underlined.

One of the defining features of the APL language is the ability to denote numeric vectors directly through juxtaposition — separating the elements by spaces, as in `0 1 1 2 3 5 8`. The notation for character “vectors” is similar to that for “strings” in most other languages, using quotes to denote the start and end of a list of characters. When generalised arrays were added to the language in the early 1980s, the most popular APL dialects extended the vector notation to allow nested arrays to be written using so-called *strand notation*, allowing the juxtaposition of sub-expressions producing arrays to form a one-dimensional array — as in `(2+2) (F00 42) MAT`

Technical specification

The notation consists of syntax that was invalid in every mainstream APL implementation before its introduction, thus causing no issues for backwards compatibility. The added syntax consists of these constructs that are currently SYNTAX ERRORS:

- *broken* round parentheses/parenthesised *name-value pair*: `( ... )`
- *broken* square brackets: `[ ... ]`
- empty round parentheses: `()`

where *broken* means interrupted by one or more statement separators (diamonds ♦ or line breaks).

- A statement here means a value expression, optionally separated with a colon `( : )` from a preceding valid APL identifier.
- Empty statements make parentheses and brackets *broken*, but do not otherwise influence the result.
- A *broken* round parenthesis creates a namespace if every* separated statement is a *name-value pair*; every such pair defines a member of the resulting

Formal syntax

The array notation can be described using Extended Backus-Naur form, where an `expression` is any traditional APL expression:

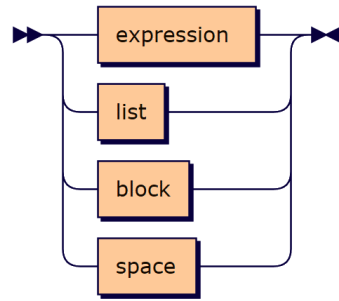
```
value ::= expression | list | block | space
list  ::= '(' ( ( value sep )+ value? | ( sep value )+ sep? ) ')'
block ::= '[' ( ( value sep )+ value? | ( sep value )+ sep? ) ']'
space ::= '(' sep? ( name ':' value ( sep name ':' value )* )? sep? ')'
sep    ::= [⋄#x000A#x000D#x0085]+
```

The list of `sep` values is for illustration purposes and is to match the line breaks recognised by the APL implementation. However, these three `sep` values should be handled when reading Unicode text files.

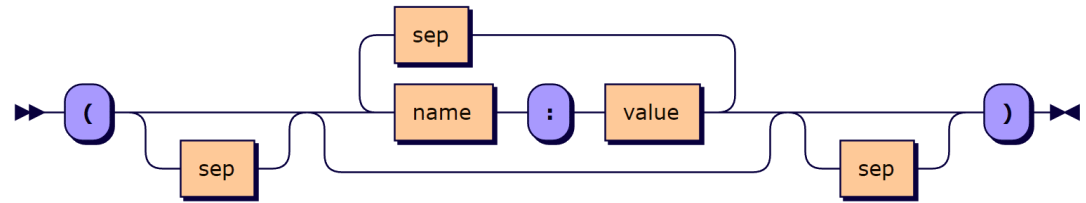
Specification

abrudz.github.io/aplan

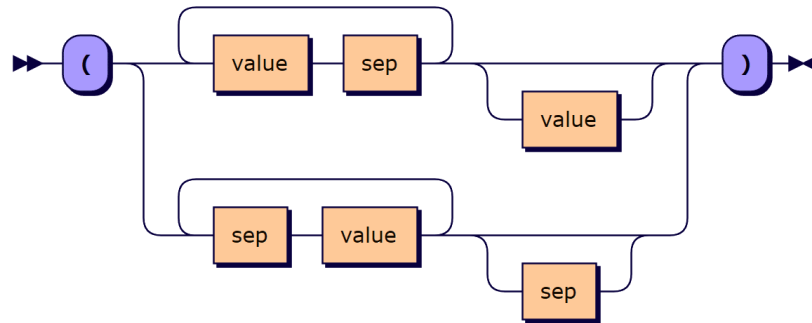
value:



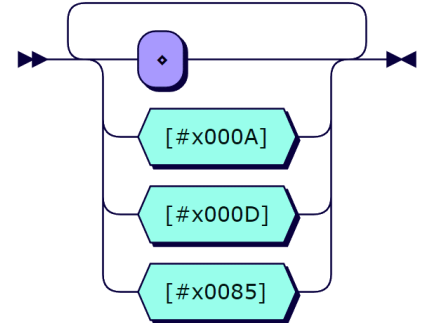
space:



list:



sep:



APL Array Notation

DYALOG

Implementation youtu.be/4cEqsBRMdW0

```
Parse←{  
  0≠ω: ' '  
  bot←0=α  
  (2≤≠ω)>v/-1↓bot:α SubParse ω  
  p←bot×SepMask ω  
  v/p:∈{1≠ω: ', <', ω ♦ ω}α(Paren ∇)EachNonempty  
  p←2(1,>/v-1↓0,</)bot  
  v/1↓p:∈(p<α)∇''p<ω  
  ω  
}
```

Implementation youtu.be/ToQoJ1Bbpus

```
parseloop: while (sp) switch(pop()) {  
    case ST_BEG: {  
        DEBUG;  
        // Initial state:  
        // [ => go to ST_LB  
        // ( => go to ST_LP  
        // ANY* => go to ST_EXPR  
        if (tok.type == TOKLB) { push(ST_LB); break; }  
        if (tok.type == TOKLP) { push(ST_LP); break; }  
        if (tok.type == TOKANY || tok.type == TOKSTR || tok.type == TOKRBR) {  
            errorxf(ESYNTAX, DMXAPLAN_UNEXPTOK, tok.offset, tok.text);  
            continue; }  
    }  
}
```



Kamila Szewczyk

Implementation

youtu.be/bSDStXeKsos

```
do
{
  sp=KwNext(sp);
  affirm(sp[1]==KEYWORD);
  switch (*sp)
  {
    case KwANDiamond: count++; break;
    case KwANNewLine: count++; break;
    case KwANCloseVector: count++; break;
    case KwANCloseMix: count++; break;
    case KwANAssign: break;
  }
} while (sp!=start);
```



John Daintree

Standardisation?

abrudz.github.io/aplan

Experimental implementations using APL models are available within some tools in the Dyalog eco-system, such as the Link tool which supports the representation of code and data in Unicode text files and the functions `Serialise` and `Deserialise` within the namespace `⎕SE.Dyalog.Array`. You can also try it out in [the interactive online sandbox](#).

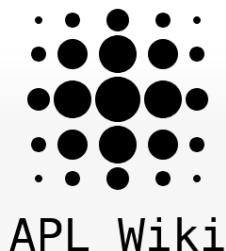
Providing feedback

Dyalog is keen to have feedback from the array language community on the notation proposed here so that we can feel confident about the design, before we proceed with our implementation. Our hope is that we will be able to keep the differences between future array notations within the family of array languages to a minimum.

We will monitor the APL Orchard chatroom, the APL Farm's #apl channel, the r/apl and r/apljk subreddits, and the comp.lang.apl newsgroup for feedback. (See [APL Wiki](#) for information about these forums.) In addition, we have created [a topic in our own forum](#). If you prefer not to comment in public, please send comments [by e-mail](#). Dyalog will update [the discussion page](#) for APL Wiki's Array notation design considerations article to contain a record of significant feedback.

Standardisation?

apl.wiki/aplan



[Main page](#)
[Recent changes](#)
[Random page](#)
[Categories](#)
[Help about MediaWiki](#)

Quick links

[Overview](#)
[Running APL](#)
[Learning resources](#)
[Chat rooms and forums](#)

Tools

[What links here](#)
[Related changes](#)

[Adám Brudzewsky](#) [Talk](#) [Dark mode](#) [Preferences](#) [Watchlist](#) [Contributions](#) [Log out](#)

Page

[Discussion](#)

Read

[Edit](#)

[View history](#)



More ▾

Search APL Wiki



Array notation

Array notation (◇), [◇]), abbreviated **APLAN** parallel to [JSON^w](#), is a way to write most [arrays](#) literally, with no or minimal use of [primitive functions](#), possibly over multiple code lines. It differs from the [strand notation](#) existing since [APL360](#) in that it can be used to write arrays of rank greater than one. Array notation is supported in [dzaima/APL](#), [BQN](#) (using angle brackets ◇) instead of round parentheses (◇)), and some tools for [Dyalog APL](#), where it is planned as an eventual language feature.



Array notation generally consists of a vector notation written with parentheses (), roughly equivalent to stranding, and a high-rank notation using square brackets [], indicating the [Mix](#) of a vector. It also supports [namespaces](#), using name:value syntax in round parentheses. [Statement separators](#) must appear between elements and between [name–value pairs^w](#).

CLEAR WS (#) -20.0u64 Link 4.1 .NET 4.8.0 - [MatPower]

File Edit View Window Session Log Action Options Tools Threads Layout Help

WS [Icons] Object [Icons] Tool [Icons] Edit [Icons] Session [Icons] APL385 Unicode 50 [Icons]

[Icons]

```
[
  [
    3 1 4
    2 7 1
    1 6 1
  ] MatPower 3

[
  130 355 111
  193 516 157
  157 425 124
]
```

```
r←m MatPower e
r←m+.×*e←[1 0 0
           0 1 0
           0 0 1]
```

Debugger Ready... CurObj: MatPower (Function)

Modified Function Last saved by: adam: 04 July 2025 10:15 Pos: 3/4,9

Ins NUM [Icons] &:1 [Icons] [Icons] [Icons] [Icons] [Icons]

Take-aways?

- Good: Lack of standards committee
- Bad: Lack of standard for syntax
- Ugly: Lack of standardisation for result

Want more APL? dyalog.com/getting-started.htm

The screenshot shows the Dyalog website homepage. The header features the Dyalog logo, the tagline "The tool of thought for software solutions", and a "Contact Us" link. Below the header is a navigation bar with links to Home, Business, Learning, Community, Resources, News, and About Us. A search bar is located on the right side of the navigation bar. The main content area has a background image of hot air balloons over a landscape. A dark blue overlay box is positioned in the center-right of the page, containing the text "Cash prizes: APL Challenge APL Forge".

DYALOG The tool of thought for software solutions

Contact Us

Blog Instagram Medium Facebook LinkedIn YouTube

Home Business Learning Community Resources News About Us

Products
Dyalog Services
Prices and Licences
Support (DSS)
Download Dyalog – Free
Dyalog '24

Home >> Learning >> Getting Started

Getting Started

Getting started with any new programming language or platform ships with enough features that you can get started. The resources on this page are free of charge and aimed at APL users.

Community

APL has a thriving and enthusiastic community of users who are very happy to answer questions:

Cash prizes:

- APL Challenge
- APL Forge

Recap

- History
- APL
- Problem
- Design Iteration
- Specification
- Implementation
- Standardisation?
- Take-aways?

Resources

- apl.wiki
- dyalog.tv
- abrudz.github.io/aplan
- [dyalog.com/
getting-started.htm](http://dyalog.com/getting-started.htm)
- apl.chat (Stack Exchange)
- adam@dyalog.com