

"APL as a tool of thought"

1. A selection of primitive functions

	ONE ARGUMENT		TWO ARGUMENTS	
+	no effect	+13	sum	10 + 13
-	reverse sign	- ^10	difference	1 - ^7
x	sign	x ^2 0 2	product	3 x 7
÷	inverse	÷ 3	divide	1 ÷ 3
	absolute value	^10 10	residue, remainder	3 10 11 12
[ceiling	[3 3.9	maximum	100 [101
l	floor	l 3 3.9	minimum	100 l 101
*	e to power	* 1	power	2 * 10
⊗	log (base e)	⊗ 10	log any base	2 ⊗ 32
o	pi times	o 2	trigon. functions	1 o 0.5
			(left arg. is 1 for sin, 2 for cos, 3 for tan)	
!	factorial	! 5	(right arg. in radians)	
?	random integer	? 100	binomial coeff.	3 ! 5
~	Boolean negation	~ 0 1 1	different random integers	7 ? 10
v			Boolean OR	0 0 1 1 v 0 1 0 1
^			Boolean AND	0 0 1 1 ^ 0 1 0 1
<			Less	3 < 5
≤			Less or equal	4 ≤ 4 5
=			Equal	'a' = 'A'
≥			Greater or equal	3 ≥ 3 2
>			Greater	3 > 12
≠			Different	1 ≠ 7×1+7
ρ	shape of array	ρ 'abc'	fill shape with	5 10ρ'ABC'
,	force vector shape	,2 2ρ'abcd'	catenate along axis	TABA,[2] TABB 'ABC','012'
φ	reverse indices (on first axis)	φ TAB	rotate	2φ'MILANO' 3 1 2φ[1] TAB
⊖	transpose	⊖ TAB	reorder axes	2 1 ⊖ TAB
⊙	reverse indices (on last axis)	⊙ TAB	rotate	3 ^-1 ⊙ TAB 2 ⊙ [2] TAB
†			take first	3†'0123456'
‡			drop first	2 ^-3 ‡ TAB
∈			belongs to	'achab'∈'abc'
⊆			find starting position	4 5 6 ⊆ 110
/			repeat	3 2/'xy' 0 1 1 0/14
\			expand	0 1 1 0\7 8 1 0 1\2]TAB
c	enclose, make it a scalar	ρ c'ABC'		
⊃	disclose, open a nested array	⊃NESTED		
ι	integers up to	ι10	index of	'abc'ι'achab'
▽	indices to sort (decr.)	▽ 20 10 40 10		
▲	indices to sort (incr.)	▲ 20 10 40 10		
⊘	execute char string	⊘ '20 x 5'		
⊕		⊕ 'A + ρB'		
⌘	format of num. array	⌘2.00 3	assigned format	10 2⌘ 1÷3
τ			number in any radix	24 60τ255 2 2 2 2τ14
⊥			reduce to base	2 2 2 2⊥0 1 1 0
⊠	matrix inverse	⊠ SQMATR	matrix division	MATRA ⊠ MATRB

2. Special syntax

()	force order of execution	(2 × 3) + 5
[;:]	indices of an array	VECTOR[3],TAB[2;1],CUBE[1;1;5] TAB[;2] , VECTOR[2 3 1 1 5]
←	assign to a name	TOT ← +/ 3 4 5 VECTOR[3+J] ← 0

→	branch to label (in function)	→SKIP
		→(X≠Y)/SKIP
		→0 exit this function
		→ exit all executions
:	label (in function)	SKIP: 'WE CONTINUE'
␣	comment (to end of line)	␣ this will not work for X>ρY
□	get (evaluated) input	3.14159 × □
	send to screen	□ ← A ← ÷ □ ← ? 10 3 ρ 10
-	prefix of negative constant	-10 -2E-6 1.1E3
◇	statement separator	A ← 1 ◇ B ← 2*10 ◇ 'DONE!'

3. Function extensions ('APL operators')

(Examples for primitives + ×)

```
10 + 10
10 20 30 + 1 2 3
TAB A + 1000
+/ TAB
+/[3] CUBE
+\ TAB
+\ [1] TAB
(110) ◦.× 110
TAB A +.× TAB B
```

4. Arrays of arrays - Strand notation

Arrays of arrays (and primitives such as < > ...) can be omitted on a first pass. It is debatable if they provide as much value-for-effort as simple arrays. But at some stage some careful exploration of arrays of arrays becomes necessary.

The traditional notation is based on < >
(< wraps an array to make it a scalar, > unwraps it)

```
NEST ← (<'ABC'), (<5 5 ρ 15), 'Z'
NEST[3] ← <1 4 ρ 'WXYZ'
(>NEST[1])[2]
```

A different notation, called strand notation, is also implemented. The same variable NEST can be assigned as

```
NEST ← 'ABC' (5 5 ρ 15) (1 4 ρ 'WXYZ')
```

The parentheses group objects, instead of forcing the order of execution.

```
( A B C ) ← NEST
```

Without these extensions, one would expect two of these forms to be errors (if X is numeric):

```
'result is ' X                    (syntax violated)
'result is ' ,X                    (mixing char and number)
'result is ' ,*X                   (safe traditional form, creates vector of char)
```

*DYALOG APL version 8
System variables - system functions*

*(Items marked ** are basic, items marked * are advanced, all others can be skipped)*

Functions, Operators & Namespaces:

AT attributes
** *CR* canonical representation
 CS change (name)space
 CY copy objects to active ws
* *ED* edit objects
* *EX* expunge objects
 EXPORT export objects
** *FX* fix definition
* *LOCK* lock object
* *ML* migration level
* *NC* name class
* *NL* name list
 NR nested representation
 NS create namespace
 NSI namespace indicator
 OR object representation
 PATH search path
** *SIZE* size of objects
* *VFI* verify & fix numerics
 VR vector representation

Input/Output:

* character input/output
** evaluated input/output
 A underscored alphabet
 A alphabet
 ARBIN arbitrary input
 ARBOUT arbitrary output
** *AV* atomic vector
 D digits
 DL delay
 FMT format
 FMT(Dyadic) format by specification
 RTL response time limit
 SD screen dimensions

 SM screen map
 SR screen read
 TC terminal control characters

APL Files:

FAPPEND append component
 FAVAIL file system availability
 FCREATE create file
 FDROP drop components
 FERASE erase file
 FHOLD hold file
 FLIB file library list
 FNAMES tied file names
 FNUMS tied file numbers
 FRDAC read file access matrix
 FRDCI read file component information

 FREAD read component
 FRENAME rename file
 FREPLACE replace component
 FRESIZE resize file
 FSIZE file size
 FSTAC set file access matrix
 FSTIE share tie file
 FTIE tie file
 FUNTIE untie files
 XT query external variable
 XT(Dyadic) set external variable

Native Files:

- * DR data representation (monadic)
- * DR data representation (dyadic)
- * NAPPEND appends value to a native file
- * NCREATE creates a native file
- * NERASE erases a native file
- * NAMES reports names of all open native files
- * NNUMS reports tie numbers of all open native files
- * NREAD reads data from a native file
- * NRENAME renames a native file

- * NREPLACE writes data to a native file, overwriting existing data
- * NRESIZE changes the size of a native file
- * NSIZE reports the current size of a native file
- * NTIE opens a native file
- * NUNTIE closes one or more native files
- * NXLATE specifies/reports character translation for a native file

Graphical User Interface:

- DQ await & process (dequeue) events
- NQ place event in queue (enqueue)
- SE session namespace
- WC create windows object
- WG get windows object properties
- WN query child object names
- WS set windows object properties

Event Trapping:

- * DM diagnostic message of last error
- EM event messages
- EN event numbers
- SIGNAL signal event
- * TRAP trap error events

Shared Variables:

- SVC query access control
- SVC(Dyadic) set access control
- SVO query degree of coupling
- SVO(Dyadic) shared variable offer
- SVQ shared variable query
- SVR retract offer
- SVS shared variable state

State Indicator Stack:

- * LC line count
- MONITOR query monitor
- MONITOR(Dyadic) set monitor
- NSI namespace indicator
- REFS local references
- SHADOW shadow names
- * SI state indicator
- STACK state indicator stack
- STATE state of an object
- STOP query stop vector
- STOP(Dyadic) set stop vector
- TRACE query trace vector

- TRACE(Dyadic) set trace vector

Workspace:

- * AI account information
- AN account name
- CLEAR clear workspace
- * CT comparison tolerance
- DIV division method
- IO index origin
- KL key label
- LOAD load workspace
- * LX latent expression
- PFKEY programmable function key
- * PP print precision
- * PW page width
- * RL random link

- SAVE save workspace

```

**      □WA workspace available
        □WSID workspace identification

System Operations:

□CMD execute command
□CMD(Dyadic) start auxiliary processor
□NA name association
□OFF sign off
□SH execute command
□SH(Dyadic) start auxiliary processor
*      □TS time stamp

```

SYSTEM COMMANDS (immediate execution only)

A *WORKSPACE* is a file containing variables and defined functions.
A single workspace is active at a given time.

Object Commands:

```

**      )COPY {wsid {names}}          )COPY UTILITY A B xy
        (copies A B xy into active workspace)

)CS {name}
**      )ED names                    )ED POLPROD (editor)
*      )FNS {name}                  )FNS
        (lists all defined functions in active workspace)

)NS {name}
)OBS {nms}
)OPS {name}
*      )PCOPY {wsid {names}}        )PCOPY UTILITY A B xy
        (copies objects unless there is a name conflict)

*      )VARS {name}                 )VARS
        (lists all variables in active workspace)

```

Stack Commands:

```

**      )RESET                      (clears all suspended executions)
*      )SI                          (shows suspended functions)
*      )SINL                        (shows suspended functions and local names)

```

Workspace Commands:

```

**      )CLEAR                      (clears all objects in active workspace)
**      )DROP {wsid}                )DROP OLDVERSION (deletes a workspace on
file)
**      )LIB {dir}                  )LIB C:\TEMP
        (lists workspaces in directory)

**      )LOAD {wsid}                )LOAD ALGEBRA
        (loads workspace ALGEBRA and loses current workspace)

**      )SAVE {wsid}                )SAVE
        (saves current workspace to file)

**      )WSID                      )WSID SOFAR
        (queries or assigns current workspace name)

)XLOAD {wsid}

```

Operating Commands:

```

)CMD {cmd}
)CONTINUE
**      )OFF                          )OFF
        (exits APL and loses current workspace)

)SH {cmd}

```

DYALOG APL KEYBOARD

◊	1	2	3	4	5	6	7	8	9	0	+	×
Q	W	E	R	T	Y	U	I	O	P	←	→	
A	S	D	F	G	H	J	K	L	[]		
Z	X	C	V	B	N	M	,	.	/	Shift	#	

Unshifted

€	¨	-	<	≤	=	≥	>	≠	√	^	-	÷
?	w	€	ρ	~	↑	↓	ι	ο	*	≡	≠	
α	∫	∫	-	∇	Δ	ο	'	□	()		
<	>	∩	□	↓	τ		;	:	\	Shift	□	

Shifted

¨	!	∇	∇	Δ	φ	ϕ	ϑ	⊗	∞	∞	∞	□	
Q	W	E	R	T	Y	U	I	O	P	□	Δ		
A	S	D	F	G	H	J	K	L	∞	∞			
Z	X	C	V	B	N	M	∞	∞	∞	Shift	∞		

Ctrl+Shift

CapsLock on produces lowercase letters
Ctrl+N switches to the APL keyboard
Ctrl+0 switches to the US text keyboard