FUNCTION/OPERATOR SYNTAX

X Y	left and right arguments of a function/operands of an operator – any array
M N	– numeric array
I J	– integer array
A B	– Boolean array
C D	– character array
f g h	functions
⍺ ⍵	left and right arguments of a function train
NS	name or reference to namespace
[ax]	indicates functions that can have an axis specified
[ct]	indicates a dependency on ⎕CT/⎕DCT
s/v/m	indicates highest rank allowed is that of a scalar/vector/matrix

SELECTED ABBREVIATIONS

actions	⎕NQ action: 0 add to queue, 1 process immediately, 2 perform default action, 3 invoke OLE method, 4 signal ActiveX event
ax_mx	three-column matrix containing userID, aggregated file operation numbers and permission numbers
bytes	byte count
cn	component number
conargs	constructor arguments
dir|file	the name of a directory/file
etype	type of new object; one of ∇ (function/operator, the default value), ∊ (vector of character vectors), − (character matrix), ⍟ (namespace script), → (simple character vector), ○ (class script) and ∘ (interface)
name	the name of a variable, function or operator in the active workspace
nvpairs	one or more name/value pairs
object|ns	a *name* or a *ref*
pn	component file pass number
pnames	character scalar or vector containing file property names
ref	a reference to a namespace or object
regex	a Perl-Compatible Regular Expression (PCRE)
rw	read or read/write
tdno	thread number
tn	tie number for files; use 0 to generate number on tie/create
trans	transformation function or numeric codes to apply to matched expressions
type	internal data type – see TYPE CODES below

TYPE CODES

Constructed by prefixing one of the following numbers with the number of bits per element:
0 Unicode char, 1 Boolean, 2 Classic (⎕AV based) char, 3 Integer, 5 Floating point,
6 Pointer to Object or Nested Array, 7 Decimal floating point, 9 Complex.

Examples: 80 = 1-byte Unicode char, 163 = 16-bit integer, 645 = double-precision float
N.B. Pointers are reported as 326 in both 32-bit and 64-bit systems

NAME CLASSES (⎕NC and ⎕NL)

	2 Array	3 Functions	4 Operators	9 Spaces
.1	2.1 Variable	3.1 Traditional	4.1 Traditional	9.1 Namespace
.2	2.2 Field	3.2 dfns	4.2 dops	9.2 Instance
.3		3.3 Derived/Primitive	4.3 Derived/Primitive	
.4				9.4 Class (OO)
.5				9.5 Interface (OO)
.6	2.6 External/Shared	3.6 External		9.6 External class
.7				9.7 External interface

PRIMITIVE FUNCTIONS

SCALAR FUNCTIONS

Scalar functions are pervasive, apply item-wise and, when dyadic, respond to the axis operator

MONADIC

Syntax	Result	Implicit Args
+Y	Conjugate ('Identity' if Y not complex)	
-N	Negate: 0-N	
×N	Direction ('Signum' if Y not complex)	
÷N	Reciprocal: 1÷N	⎕DIV
⌊N	Round down to integer	[ct]
⌈N	Round up to integer	[ct]
|N	Magnitude (absolute value)	
*N	e raised to the power N	
⍟N	Natural logarithm of N	
○N	pi times N	
!N	Factorial (Gamma function of N+1)	
?J	Random number selected from ⍳J (when J=0, a real number from <0,1>)	⎕IO, ⎕RL
~B	Logical Inverse: 0=B	

DYADIC

Syntax	Result	Implicit Args
M+N	Add N to M	
M-N	Subtract N from M	
M×N	Multiply M and N	
M÷N	Divide M by N	⎕DIV
M|N	Residue after dividing N by M	[ct]
M*N	M raised to the power N	
M⍟N	Base-M logarithm of N	
M⌈N	Maximum of M and N	
M⌊N	Minimum of M and N	
I○N	Circular functions[1]	
M!N	Number of selections of size M from N (Beta fn)	
M∧N	Lowest Common Multiple of M and N	[ct]
M∨N	Greatest Common Divisor of M and N	[ct]
< ≤ ≥ >	Numeric comparisons[2]	[ct]
= ≠	General comparisons[2]	[ct]
∧ ∨ ⍲ ⍱	Boolean functions[3]	

[1] Circular functions (angles in radians)

(-Is)○N	Is	Is○N
(1-N*2)*.5	0	(1-N*2)*.5
Arcsin N	1	Sin N
Arccos N	2	Cos N
Arctan N	3	Tan N
(N+1)×((N-1)÷N+1)*.5	4	(1+N*2)*.5
Arcsinh N	5	Sinh N
Arccosh N	6	Cosh N
Arctanh N	7	Tanh N
-8○N	8	(-1+N*2)*.5
N	9	<real N>
+N	10	|N
N×0J1	11	<imaginary N>
*N×0J1	12	<phase of N>

[2] Comparisons

Comparisons return:
- 1 if proposition is true
- 0 if proposition is false

[3] Boolean functions

A←1 0 0 1
B←1 0 1 0

A∧B	1	0	0	0
A∨B	1	0	1	1
A⍲B	0	1	1	1
A⍱B	0	1	0	0
~B	0	1	0	1

PRIMITIVE FUNCTIONS continued

NON-SCALAR FUNCTIONS

NON-SCALAR MATHEMATICAL

Syntax	Result	Implicit Args
⌹Nm	Matrix inverse of Nm (square Nm)	
⌹Nm	Matrix pseudo-inverse of Nm (over-determined Nm)	
Mm⌹Nm	Multiply Mm with inverse of Nm	
M⊤N	Encode value N in number system M	
M⊥N	Decode: Evaluate N in number system M	

ARRAY PROPERTIES

Syntax	Result	Implicit Args
⍴Y	Shape: Length of each axis of Y	
≡Y	Depth: Maximum level of nesting in Y (-ve if uneven)	⎕ML
≢Y	Tally: Number of items in leading axis	

STRUCTURAL

Change structure, typically keeping all items

Syntax	Result	Implicit Args
⊂Y	Enclose: Scalar containing Y	[ax]
⊆Y	Nest: Y if already nested, else scalar containing Y	
↑Y	Mix: Remove nesting (⎕ML 1)	⎕ML, [ax]
↓Y	Split: Nest sub-arrays	[ax]
∊Y	Enlist: Simple vector from elements of Y (⎕ML 1)	⎕ML
,Y	Ravel: Reshape into a vector	[ax]
⍪Y	Table: Reshape into 2-dimensional array	
⌽Y	Reverse last axis of Y	[ax]
⊖Y	Reverse leading axis of Y	[ax]
⍉Y	Transpose: Reverse order of axes of Y	
Iv⍴Y	Reshape Y to have shape Iv	
I⌽Y	Rotate vectors along last axis of Y	[ax]
I⊖Y	Rotate vectors along leading axis of Y	[ax]
Iv⍉Y	Reorder the axes of Y	⎕IO
X,Y	Catenate: Join along last axis	[ax]
X⍪Y	Catenate First: Join along leading axis	[ax]

INDEX GENERATORS

Syntax	Result	Implicit Args
⍳Jv	Indices of all items of array of shape Jv	⎕IO
⍸B	Indices of all 1s in B	⎕IO
⍋Y	Upgrade: Indices to reorder Y ascending	⎕IO
⍒Y	Downgrade: Indices to reorder Y descending	⎕IO
X⍳Y	Index of: Indices in X of items of Y	⎕IO, [ct]
X⍸Y	Indices of items of Y in intervals with cut-offs X	⎕IO
Is?Js	Deal: Is distinct items from ⍳Js	⎕IO, ⎕RL
C⍋D	Upgrade using collation sequence C	⎕IO
C⍒D	Downgrade using collation sequence C	⎕IO

SET FUNCTIONS

Syntax	Result	Implicit Args
∪Yv	Unique: Distinct items of Yv	[ct]
X∊Y	For each item of X, 1 if found in Y, else 0	[ct]
X⍷Y	Occurrences of entire array X within Y	[ct]
X≡Y	Match: 1 if X is identical to Y, else 0	[ct]
X≢Y	Not Match: ~X≡Y	[ct]
Xv~Y	Without: (~Xv∊Y)/Xv	[ct]
Xv∪Yv	Union: Xv,Yv~Xv	[ct]
Xv∩Yv	Intersection: (Xv∊Yv)/Xv	[ct]

PRIMITIVE FUNCTIONS continued

SELECTION

Select items from an array

Syntax	Result	Implicit Args
⊃Y	First item of Y (⎕ML 1)	⎕ML, [ax]
Iv⊃Y	Reach into Y along path given by Iv	⎕IO
Iv⌷Y	Index Y using indices Iv	⎕IO, [ax]
Iv↑Y	Take Iv items along axes of Y	[ax]
Iv↓Y	Drop Iv items along axes of Y	[ax]
Iv/Y	Replicate along last axis of Y	[ax]
Iv⌿Y	Replicate along leading axis of Y	[ax]
Iv\Y	Expand last axis of Y	[ax]
Iv⍀Y	Expand leading axis of Y	[ax]
Av⊂Y	Partitioned enclose of Y according to Av (⎕ML 1)	⎕ML, [ax]
Mv⊆Y	Partition Y according to Mv	[ax]

DATA CONVERSION

Syntax	Result	Implicit Args
⍎Dv	Execute: Result of expression Dv	
⍕Y	Format: Character representation of Y	
NS⍎Dv	Execute Dv within namespace NS	
Iv⍕Y	Format Y using (width, decimals) pairs Iv	

IDENTITY FUNCTIONS

Return an argument unchanged

Syntax	Result	Implicit Args
⌷Y	Materialise items of Y in workspace	
⊣Y	Same: Y	
⊢Y	Same: Y	
X⊣Y	Left: X	
X⊢Y	Right: Y	

DFN SYNTAX

{⍺ function ⍵}		{⍺⍺ operator ⍵⍵}			
⍺	left argument	⍺⍺	left operand	:	guard
⍵	right argument	⍵⍵	right operand	::	error guard
∇	self reference	∇∇	self reference	⍺←	default left argument
				1:s←	shy result

FUNCTION TRAINS

```
 ( gh)⍵ →        g( h⍵)  ⍝ monadic atop
⍺( gh)⍵ →        g(⍺h⍵)  ⍝ dyadic  atop

 (fgh)⍵ → ( f⍵)  g( h⍵)  ⍝ monadic fgh fork
⍺(fgh)⍵ → (⍺f⍵)  g(⍺h⍵)  ⍝ dyadic  fgh fork

 (Xgh)⍵ →       Xg( h⍵)  ⍝ monadic Xgh fork
⍺(Xgh)⍵ →       Xg(⍺h⍵)  ⍝ dyadic  Xgh fork
```

PRIMITIVE OPERATORS

MONADIC

Syntax	Result
{Is}f/Y	Reduce: f between all items of Y (in groups of Is) on last axis
{Is}f⌿Y	Reduce First: f between all items of Y (in groups of Is) on first axis
f\Y	Scan: f between items of Y in progressively longer vectors along last axis
f⍀Y	Scan First: f between items of Y in progressively longer vectors along first axis
{X}f¨Y	Each: f on items of Y or between items of X and Y
Xf⌸Y	Key: f on items of Y grouped by unique X values
f⌸Y	Key: f on first axis indices of Y grouped by unique Y values
{X}f⍨Y	Commute: same as YfX (or YfY if no X specified)
{X}f&Y	Spawn: f on Y (or between X and Y) in a new thread
{X}(Ns⌶)Y	I-beam: Call experimental system-related service Ns

DYADIC

Syntax	Result
{X}(f⍤r)Y	Rank: f on or between trailing rank-r subarrays
(f⌺Jm)Y	Stencil: f on (possibly overlapping) rectangles of Y
{X}(f⍣g)Y	Power: iterates f (or X∘f) on Y until condition YgfY (or YgXfY) is true
{X}(f⍣Js)Y	Power: f (or X∘f) on Y Js times
Xf.gY	Inner Product: f / g between trailing vectors of X and leading vectors of Y
X∘.gY	Outer Product: g between each item of X and every item of Y
f∘gY	Compose (I): f on the result of g on Y, that is, fgY
Xf∘gY	Compose (IV): X∘f on the result of g on Y, that is, XfgY
X∘gY	Compose (II): g between X and Y, that is, XgY
(f∘Y_2)Y_1	Compose (III): f between Y_1 and Y_2, that is, Y_1fY_2
{X}(f⍠Zv)Y	Variant: f qualified by Zv on Y (or between X and Y)
(X@N)Y	At: use values in X to replace positions N in Y
{X}(f@N)Y	At: apply f (or X∘f) to modify positions N in Y
(X@g)Y	At use values in X to replace positions identified by Boolean mask (gY) in Y
{X}(f@g)Y	At: apply f (or X∘f) to modify positions identified by Boolean mask (gY) in Y

CONTROL STRUCTURES

```
:For var :In|:InEach ax ⋄ block ⋄ :EndFor
:Hold tkn ⋄ block ⋄ :Else ⋄ block ⋄ :EndHold
:If bx ⋄ block ⋄ :ElseIf bx|:Else ⋄ block ⋄ :EndIf
:Repeat ⋄ block ⋄ :Until bx ⋄ :AndIf bx|:OrIf bx
:Repeat ⋄ block ⋄ :EndRepeat
:Select ax ⋄ :Case val|:CaseList val ⋄ block ⋄ :Else ⋄
                                          block ⋄ :EndSelect
:Trap ecode ⋄ block ⋄ :Case ecode|:CaseList ecode ⋄ block ⋄
                                    :Else ⋄ block ⋄ :EndTrap
:While bx ⋄ block ⋄ :AndIf bx|:OrIf bx ⋄ block ⋄ :EndWhile
:While bx ⋄ block ⋄ :AndIf bx|:OrIf bx ⋄ block ⋄ :Until bx
:With ns ⋄ block ⋄ :EndWith
```

block	one or more APL statements to be executed
ax	an expression returning an array
bx	an expression returning a single Boolean value (0 or 1)
ecode	an integer scalar or vector containing the list of event codes to be handled
ns	a namespace within which actions will be performed
tkn	the tokens that must be acquired before the thread can continue
val	an expression to compare with the array returned by <ax>
var	one or more loop variable name

:Continue – start next iteration of surrounding **:For**, **:Repeat** or **While**

:Leave – terminate **:For**, **:Repeat** or **While**

:Return – equivalent to →0