

APL Reference

<i>Operation</i>	<i>Expression</i>	<i>Value</i>
Data creation	<code>newint: int -> aplval</code> make a new integer value	<code>newint 1</code>
	<code>newveci: int list -> aplval</code> make a new int vector	<code>newveci [1;2;3]</code>
Sample data	<code>A = rho (newveci [2;3]) (indx (newint 6))</code>	1 2 3 4 5 6
	<code>V = newveci [2;4;6]</code>	2 4 6
	<code>C = newveci [1;0]</code>	1 0
	<code>D = newveci [1;0;1]</code>	1 0 1
Arithmetic	<code>A *@ A</code>	1 4 9 16 25 36
	<code>V -@ (newint 1)</code>	1 3 5
	<code>+@, /@, %@</code>	
Unary	<code>~@ V</code>	-2 -4 -6
Relational	<code>A >@ (newint 4)</code>	0 0 0 0 1 1
	<code>=@, <@, >=@, <=@, <>@</code>	
Logical	<code>&@, @</code>	
Reduction	<code>!+ V</code>	12
	<code>maxR A</code>	3 6
	<code>!-, !*, !/, !%, ! , !&, minR</code>	
Compression	<code>D % V</code>	2 6
	<code>C % A</code>	1 2 3 (a 1,3-matrix)
Shape	<code>shape A</code>	2 3
Ravelling	<code>ravel A</code>	1 2 3 4 5 6
	<code>ravel (newint 1)</code>	1
Restructuring	<code>rho (shape A) V</code>	2 4 6 2 4 6
	<code>rho (shape V) C</code>	1 0 1
	Note: rho requires a vector as the first argument	
Catenation	<code>A ~@ C</code>	1 2 3 4 5 6 1 0
Index generation	<code>indx (newint 5)</code>	1 2 3 4 5
Transposition	<code>trans A</code>	1 4 2 5 3 6
Subscripting	<code>V @@ (indx (newint 2))</code>	2 4
	<code>A @@ (newint 1)</code>	1 2 3 (a 1,3-matrix)
	<code>(trans A) @@ (indx (newint 2))</code>	1 4 2 5
Auxiliary	<code>show: aplval -> unit</code> displays the given value	
	<code>isTrue: aplval -> bool</code> is the given value True or False?	