J Reference Card for version 6.01

Arithmetic Dyads

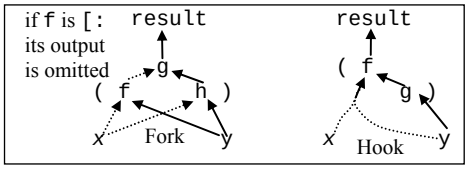
`2 + 8`	Plus	`10`
`2 - 8`	Minus	`_6`
`2 * 8`	Times	`16`
`2 % 8`	Divide	`0.25`
`2 | 8`	Residue	`0`
`2 ^ 8`	Exponent	`256`
`2 ^. 8`	Log	`3`
`3 %: 8`	Root	`2`
`2 >. 6`	Greater	`6`
`2 <. 6`	Lesser	`2`
`36 +. 24`	GCD	`12`
`36 *. 24`	LCM	`72`
`10 10 #. 8 3`	Base	`83`
`10 10 #: 83`	Antibase	`8 3`
`2 ! 8`	CombOutOf	`28`
`2 ? 8`	Deal	`4 2`

Arithmetic Monads

`<. 4.5`	Floor	`4`
`>. 4.5`	Ceiling	`5`
`* _4 0 3`	Sign	`_1 0 1`
`! 4`	Factorial	`24`
`? 20`	Random in `i. y`	`7`
`? 0`	Random in (0,1)	`0.452`

Legend

`x y`	args to verbs
`m n`	nouns
`u v`	verbs
italics	optional



Comparisons (result 1 if TRUE)

`=`	Equal	`>`	Greater	`>:`	GreaterOrEqual
`~:`	NotEqual	`<`	Less	`<:`	LessOrEqual

`-:` Match (rank _ _): equal in shape, boxing, and values; but if empty, type does not matter

`'re' E.'reread'` WindowedMatch (ranks may be > 1) `1 0 1 0 0 0`

Comparisons are tolerant; follow by `!.0` for exact comparison

Assignments

`(n) =. v`	AssignInd: value of `n` gives name(s) to assign
`'n1 n2' =. v1;v2`	AssignMult: one level of boxing is removed
`'`add sub' =. +`-`	AssignAR

Shorthands

`<: 5`	`y-1`	Decrement	`4`
`>: 5`	`y+1`	Increment	`6`
`% 5`	`1%y`	Reciprocal	`0.2`
`-. 0.3`	`1-y`	Complement	`0.7`
`+: 9`	`y*2`	Double	`18`
`-: 9`	`y%2`	Halve	`4.5`
`*: 9`	`y^2`	Square	`81`
`%:`	`2%y`	SquareRoot	`3`
`^ 1`	`e^y`	Exp	`2.718`
`^. 7.389`	`e^.y`	NaturalLog	`2`
`#. 1 0 1`	`2#.y`	FromBase2	`5`
`#: 5`	`2#:y`	ToBase2*	`1 0 1`
`m&#.^:_1`	`m#:y`	ToBasem*	

*ToBase2 and ToBasem produce as many result items as needed to hold significance

Searches

`'people' i. 'pow'`	IndexOf[1]	`0 2 6`
`'people' i: 'pow'`	IndexOfLast[1]	`3 2 6`
`'pow' e. 'people'`	ElementOf[1]	`1 1 0`
`I. 0 1 1 0 1`	IndicesOfOnes	`1 2 4`
`0 2 4 I. 2 3 _1 9`	FindInsertionPoint[12]	`1 2 0 3`
`(i. >./) 1 2 8 5`	IndexOfLargest[34]	`2`
`3 (= i. 1:) 1 3 3 0`	FindFirstTrue[456]	`1`
`3 ([: I. =) 1 3 3 0`	IndicesWhereTrue[67]	`1 2`
`m&i.  e.&n  m&i:`	FastSearch (when used repeatedly)	

[1] rank searched for is rank of items of other operand [2] min index before which item can be inserted in order [3] or `<.` [4] or `i:` [5] or `0:` [6] any comp. or `e.` [7] or `+/ +./ *./`

Operations on Ordered Sets

`'rare'-. 'er'`	RemoveItems	`a`
`~. 'rare'`	UniqueItems	`rae`
`~: 'rare'`	UniqueSieve	`1 1 0 1`
`i.~ 'rare'`	SelfClassify	`0 1 0 3`

Selections

A is `abcd efgh ijkl mnop`

`1 0 2 # 'abc'`	Copy	`acc`
`1j1 0 2 # 'abc'`	CopyFill	`a cc`
`1j1 0 2 #!.'*' 'abc'`	CopyCustom	`a*cc`
`1 0 1&#^:_1 'ab'`	Expand	`a b`
`1 0 1&#^:_1!.'*' 'ab'`	ExpandCustom	`a*b`
`_1 1 { A`	ItemsFrom	`mnop efgh`
`1 3 {"1 A`	FromEachRow	`bd fh jl np`
`2 1 { A`	From (All axes scalar)	`j`
`1 3 { A`	From (Omitted trailing axis)	`efgh mnop`
`1 2 { A`	From (Axis 1 complementary)	`efh`
`1 3 1 0 2 { A`	From (General axes)	`feg nmo`
`(<a:;2 0) { A`	From (Omitted early axis)	`ca ge ki om`
`1 1 / 3 2 (<"1@[ { ]) A`	FromUnboxed (fast form)	`fo`

Monad i. & i:

`i. 3` / `0 1 2`
`i. _3` / `2 1 0`
`i. 2 3` / `0 1 2` / `3 4 5`
`i: 2` / `_1 0 1`

Operations on Booleans

`16b1a` Base16 constant `26`

Dyads: `+.` OR, `+:` NOR

Monad: `-.` NOT; `*.` AND, `*:` NAND, `=` XOR, `~:` XNOR

`> < >: <:` are also meaningful

`m b.` (0≤m<16) Boolean function with truth table `2 2#:4 4#:m` (`1 b.` is AND)

`m b.` (16≤m<32) bitwise Boolean; applies `(m-16) b.` to each bit of integers

`x 32 b. y` rotate `y` left `x` bits

`x 33 b. y` unsigned

`x 34 b. y` signed

shift L (x>0) or R (x<0) `x` bits

Take and Drop

`2 {. i. 6`	Take	`0 1`
`_2 {. i. 6`	TakeLast	`4 5`
`2 }. i. 6`	Drop	`2 3 4 5`
`_2 }. i. 6`	DropLast	`0 1 2 3`
`4 {. 2 3`	Overtake	`2 3 0 0`
`4 {.!.9 (2 3)`	OvertakeCustom	`2 3 9 9`
`2 _2 {. i. 4 4`	TakeMultiAxis	`2 3` / `6 7`

Monad	. First	: Last
{ Take	{. Head	{: Tail
} Drop	}. Behead	}: Curtail

apply to `0 1 2`: `0 2` / `1 2 0 1`

Box Operations

B is `0 1 | 2 3 | 4 5 | 6 7 | 8`

`L. B`	Level	`2`
`$ L:0 B`	AtLevel	`,2 ,2 ,2 ,2`
`{. L:1 B`	AtLevel	`0 1 6 8`
`# S:0 B`	Spread	`2 2 2 2 1`
`|.&.> B`	Each (fast)	`4 5 2 3 0 1 7 6 8`
`1 {:: B`	Fetch	`6 7`
`0 1 {:: B`	Fetch	`2 3`
`0 2 0 {::B`	FetchList	`4 5 0 1`

Join and Reshape

`ab` / `, cd`	Enfile	`abcd`
`'ab' , 'cd'`	Append	`abcd`
`0 1` / `2 3 , 8 9`	Append (unequal ranks)	`0 1` / `2 3` / `8 9`
`0 1` / `2 3 , ,8`	Append (short)	`0 1` / `2 3` / `8 0`
`0 1` / `2 3 , 8`	Append (atom)	`0 1` / `2 3` / `8 8`
`,. 'ab'`	EnfileItems	`a` / `b`
`'ab' ,. 'cd'`	AppendItems	`ac` / `bd`
`$ ,: 'ab'`	Itemize (adds leading axis)	`1 2`
`'ab' ,: 'cd'`	Laminate	`ab` / `cd`
`3 $ 0 1` / `2 3`	ReshapeItems	`0 1` / `2 3` / `0 1`
`3 ($,) 0 1` / `2 3`	Reshape	`0 1 2`
`3 ; (4 ; 5)`	Link	`3 4 5`
`3 ,&< (4 ; 5)`	JoinBoxed	`3 4 5`
`; 0 1` / `2 3 ,4 6`	Raze (expand items of opened boxes to size of largest, then append)	`0 1` / `2 3` / `4 0` / `6 6`
`;: '2 wds'`	JWords	`2 wds`
`;:^:_1 w1 w2`	RazeWords	`w1 w2`

Whole-Array Operations

`|. 'abcde'`	Reverse	`edcba`
`2 |. 'abcde'`	RotateLeft	`cdeab`
`_2 |. 'abcde'`	RotateRight	`deabc`
`2 |.!.'*' 'abcde'`	ShiftLeft	`cde**`
`|.!.'*' 'abcde'`	ShiftRightOne	`*abcd`
`1 _1 |.` `abcd` / `efgh` / `ijkl` / `mnop`	Rotate (multiaxis)	`hefg` / `lijk` / `pmno` / `dabc`
`|:` `abcd` / `efgh` / `ijkl` / `mnop`	Transpose (reverse axes)	`aeim` / `bfjn` / `cgko` / `dhlp`
`x |: y`	ReorderAxes (moves axes x to end of axes)	
`'c0 c1 c2' =. |: y`	AssignIndividualColumns	
`/: 3 1 4 1`	GradeUp*	`1 3 0 2`
`/:~ 3 1 4 1`	SortUp*	`1 1 3 4`
`'abcd' /: 3 1 4 1`	SortUpUsing*	`bdac`
`/:@/: 3 1 4 1`	Ordinals*	`2 0 3 1`
`'*' (<1 2)}` `abcd` / `efgh` / `ijkl` / `mnop`	Amend	`abcd` / `ef*h` / `ijkl` / `mnop`
`'*+' [`(#@[)`]}` `ab` / `ef` / `ij` / `mn`	Amend (gerund form)	`ab` / `ef` / `*+` / `mn`
`y =. x m} y`	AmendInPlace (fast form)	

* use `\:` for descending ordering

Partitions

`<\ i. 3`	Prefixes	`|,0|0 1|0 1 2|`
`2 <\ i. 4`	Infixes	`|0 1|1 2|2 3|`
`_2 <\ i. 5`	Infixes, no overlap	`|0 1|2 3|,4|`
`<\. i. 3`	Suffixes	`|0 1 2|1 2|,2|`
`3 <\. i. 5`	Outfixes	`|3 4|0 4|0 1|`
`_3 <\. i. 5`	Outfixes, no overlap	`|3 4|0 1 2|`
`3 4` `2 2` `u;.3`	**u** applied to SubArrays[1] (**bold** and ***italic***)	**ab**cd**ef**gh **ij**kl**mn**op qrstuvwx **yx**01**23**45 **67**89**AB**CD EFGHIJKL ***MN***OP***QR***ST
`3 4` `2 2` `u;._3`	**u** applied to FullSubArrays[1] (**bold**)	abcdef ghi**jk**l mno**pq**r stu**vw**x
`1 _2` `3 2` `u;.0`	**u** applied to SubArray[2] (**bold**)	
`<;.1 'people'`	CutOnHead[3]	`|peo|ple|`
`0 1 0 1 <;.1 i. 4`	CutStartAtOne[3]	`|1 2|,3|`
`<;.2 'people'`	CutOnTail[3]	`|pe|ople|`
`0 1 0 1 <;.2 i. 4`	CutEndAtOne[3]	`|0 1|2 3|`
`'people' <\/. i. 6`	Key	`|0 3|1 5|,2|,4|`

The operation, shown in **bold** in the examples as **<** or **u**, can be replaced by any verb, or with a gerund `m` in which case the components of `m` are applied cyclically, one per partition.

[1] `x` is *`boundary,:`*`shape`. Subarrays start at all possible combinations of multiples of the atoms of `boundary`, and have the shape `|shape`. A negative component of `shape` reverses that axis in each subarray.

[2] `x` is *`corner,:`*`shape`. The subarray starts at `corner` and has shape `|shape`. A negative component of `corner` causes the subarray to extend backward in that component; a negative component of `shape` reverses that axis in the subarray.

[3] `;._1` omits the first, and `;._2` the last, item in each partition.

Insert

`u/ y`	Insert `u` between items of `y`	
`u/ 1 3 5`	Insert	`1 u 3 u 5`
`+/ 1 3 5`	Sum	`9`
`+/\ 1 3 5`	RunningSum	`1 4 9`
`+/\. 1 3 5`	RevRunningSum	`9 8 5`
`m/ y`	Insert verbs from gerund `m`	

Gerunds

`u` ` v`	TwoVerbGerund	
`u` ` ''`	OneVerbGerund	
`+:` ` *: ` ` :0 i. 3`	Append verb results	`0 2 4` `0 1 2`
`+` ` '' ` ` :6`	MakeVerb	`+`
`]` ` !` ` -@.* 0 3 _2`	Agenda*	`0 6 2`

*`u@.v` (rank `v`) is *`x`* `((`*`x v y`*`){u)` ` :6 y`

Control Structures

```
if. T do. B0 else. B1 end.
if. T do. B0 elseif. T1 do. B1 elseif. T2 do. B2 end.[1]
while. T do. B end.[1]        whilst. T do. B end.[1] (skips T first tme)
for. T do. B end. loop #T times    for_xyz. T do. B end.[2]
    break. jump out of loop    continue. go to end of loop
select. T fcase. T0 do. B0 fcase T1 do. B1 end.[1] (fcase falls through)
try. B0 catch. B1 catcht. B2 end.  execute B1 if error in B0[3]
returncode return.
```

[1] omitted T is true [2] sets *`xyz`* and *`xyz`*`_index` for each loop [3] `catcht.` catches `throw.` from a called function

Shape and Rank

`$ i. 2 3`	ShapeOf	`2 3`
`# i. 2 3`	TallyOf	`2`
`#@$ i. 2 3`	RankOf	`2`
`+/ 0 1` `2 3`		`2 4`
`+/"1 0 1` `2 3`		`1 5`
`+/"0 0 1` `2 3`		`0 1` `2 3`
`1 2 +"0 0 1 2` `3 4 5`		`1 2 3` `5 6 7`
`1 2 3 +"1 0 1 2` `3 4 5`		`1 3 5` `4 6 8`
`1 2 3 +"0 0 1 2` `3 4 5`		`length error`

`x u/ y` applies `u` between each cell of `x` and all of `y`

Trigonometry and Calculus

`1 o. 1r3p1`	Sin	`0.866`
`2 o. 1r3p1`	Cos	`0.5`
`3 o. 1r3p1`	Tan	`1.732`
`other o. y`	Trig Functions	
`o. 1`	PiTimes	`3.1416`
`p. 6 5 1`	Roots	`|1|_3 _2|`
`p. |1|_3 _2|`	Coeffs	`12 10 2`
`1 2 1 p. 2`	EvalPoly	`9`
`|1|_1 _1| p. 2`	EvalPoly	`9`
`p.. 1 2 1`	PolyDeriv	`2 2`
`9 p.. 1 2 1`	PolyIntegral	`9 1 1 0.33`
`*: d. 1`	Derivative	`+:`
`*: D. 1`	PartialDeriv	
`*:` ` +: D. 1`	AssignDeriv	
`1e_8 u D: n y`	SecantSlope of nth derivative	
`^ t. 1 2 3`	TaylorCoeff	`1 0.5 0.167`
`u` ` v t. n`	AssignTaylor	
`^ t: 1 2 3`	ExpTaylor	`1 1 1`
`^ T. 3`	TaylorApprox	`1 1 0.5&p.`
`m H. n`	HypergeometricSeries	

Format

`99 ". '2 5.5 xx'`	Numbers	`2 5.5 99`
`5j2 ": 1.468 2.3`	Format	`1.47 2.30`
`7j_2 ": 2.3`	Format	`2.30e0`
`'n[CR]4.' 8!:2 ]3 _5`	Format	`3 5CR`

Matrix Operations

`%. y`	MatrixInverse
`x %. y`	MatrixDivide
`x +/ .* y`	MatrixMultiply
`-/ .* y`	Determinant
`+/ .* y`	Permanent

Mathematics

`A. 2 0 1`	AnagramIndex	`4`
`4 A. 'abc'`	Anagram	`cab`
`C. 2 1 0`	PermForm	`|1|2 0|`
`|1|2 0| C. 'abc'`	Permute	`cba`
`C.!.2 y`	PermParity	`_1 0 1`
`p: 3`	YthPrime	`7`
`x p: y`	PrimeInfo	various
`q: 56`	PrimeFactors	`2 2 2 7`
`_ q: 56`	PrimeExps	`3 0 0 1`
`__ q: 56`	PrimeFacExp	`2 7` `3 1`
`x: 1%3`	Exact	`1r3`
`x:^:_1 (1r3)`	Inexact	`0.3333`
`2 x: 1r2`	NumDenom	`1 2`

Adverbs and Conjunctions

`u~ y`	Reflexive: `y u y`
`x u~ y`	Passive: `y u x`
`x` `u^:n y`	Power: execute *`x`*`&u` for `n` times; if `n<0`, execute inverse of *`x`*`&u` for –`n` times; if `n=0`, result is `y`
`x` `u^:v y`	Power, where `n` is given by *`x`* `v y`
`x` `u^:v y`	If: `y` if *`x`* `v y` is false (`0`), *`x`* `u y` if *`x`* `v y` is true (`1`)
`u^:_`	Converge: repeat `u` until result is constant
`x` `u^:v^:_ y`	DoWhile: repeat `u` while *`x`* `v y` is `1`
`u^:a:`	ConvergeHistory: repeat `u` until result is constant, return all intermediate values
`{~^:a:&0`	ChaseChain: follow chain of record positions
`u :.v`	Inverse: like `u`, but inverse is `v`
`u ::v`	Adverse: `u`, but execute `v` if error during `u`
`x` `u@:v y`	At: `u` *`x`* `v y`
`x` `u@v y`	Atop*: *`x`* `u@:v"mv y`
`1 2 +/@* 3 4`	`3 8` NB. `(+/ 1*3) , (+/ 2*4)`
`1 2 +/@:* 3 4`	`11` NB. `+/ 1 2 * 3 4`
`x` `u&:v y`	Appose: *`(v x)`* `u v y`
`x` `u&v y`	Compose*: *`x`* `u&:v"mv y`
`x` `u&.:v y`	Dual: `v^:_1` *`(v x)`* `u v y`
`x` `u&.v y`	Dual*: *`x`* `u&.:v"mv y`
`>:&.> 1 2 3`	`|2|3|4|`
`>:&.:> 1 2 3`	`|2 3 4|`
`m&v y` or `u&n y`	MonadFromDyad: `m v y` or `y u n`
`x` `m&v y`	same as `(m&v)^:x y` (`x u&n y` similarly)

* `mv` is monadic rank of `v`

Complex Numbers

`2x1`	ExpNum	`2*`*e*`^1`
`1p2`	CircNum	`1*`π`^2`
`+ 3j4`	Conjugate	`3j_4`
`+. 3j4`	RealImag	`3 4`
`*. 3j4`	LenAngle	`5 0.927`
`| 3j4`	Magnitude	`5`
`j. 1j2`	TimesJ	`_2j1`
`3 j. 4`	Complex	`3j4`
`r. 1r3p1`	Cis (`^j. y`)	`0.5j0.87`
`2 r. 1p1`	TimesCis	`_2j0`

Selected Foreigns & Miscellaneous

`". '2 + 3'`	`5` Execute sentence
`u b. y`	Info on `u`: `y=_1` inverse; `0` ranks; `1` identity function
`3!:0 y`	Datatype of `y`
`3!:1 y`	Binary representation of `y` as coded character string
`3!:3 y`	Binary representation of `y` as displayable hex array
`x 3!:4 inty` `x 3!:5 floaty`	Numeric/bytestring conversion. `x>0`: convert list `y` to char list, `2^x` (int) or `2^>:x` (float) chars/number. `x<0`: convert char list `y` to numeric list, `2^-x` (int) or `2^1-x` (float) chars/number. `x=0`: 2-byte short to unsigned int
`4!:0 <'name'`	Class of `name`, `_1` if undefined
`5!:5 <'name'`	String which, if interpreted, creates the value of `name`
`6!:0 ''`	Current time Y M D H M S
`x` `6!:2 'sentence'`	Time to execute `sentence` *`x`* times
`7!:2 'sentence'`	Space to execute `sentence`

`$.` Sparse matrix `$:` Recursion `s:` Symbol `u:` Unicode

Copyright © 2006 Henry H. Rich