APL and J

- Functional programming with linear algebra
- Large set of primitive operations
- Very compact program representation
- Regular grammar, uniform abbreviation rules
- Functions restricted to unary and binary forms (monadic and dyadic)
- Uniform naming rules for formals simplifies function composition
- Interactive, interpreted, dynamically typed, garbage-collected
What it looks like

- Polynomial evaluation:
  Given a vector of coefficients $C$ and a value $x$, the value of $c_0 + c_1x + ..$ is
  
  $$+/C * X \^ i. \$ C$$

  $i.$C is a vector of exponents: $0 .. n$

  $X^i.$ is the vector (1 X, $X^2.. X^{(n-1)}$

  Multiply by vector of coefficients and add
Basic components

Fundamental data-structure is n-dimensional array
Components are scalar (mechanism for nesting)
Atomic operations on scalars extend pointwise to n-dimensional structures: built-in iterators obviate the needs for conventional control structures

- **A single numeric type**: integers are floating point numbers with no fractional part
- **Booleans are arithmetic values**: arrays can serve as boolean masks
Using the syntactic space

- $a^b$: exponentiation
- $^b$: e to the power $b$
- $! a$: factorial
- $a!b$: binomial coefficient
- $a \% b$: division
- $\% b$: inverse
- $? a$: random number between 0 and $a$
- $b ? a$: vector of size $b$, of random numbers
Shape

- Arrays characterized by their dimensions:
  \[ v =: 2 \ 3 \ 5 \ 7 \ 11 \ 13 \quad \text{NB. input} \]
  \[ $v \]
  6
  \[ $$v \]
  1
  \[ $v \] is the vector of dimensions
  \[ $$v \] is the number of dimensions
  \[ $5 \]
  (nothing: 5 is a scalar)
  \[ $$5 \]
  0 \quad \text{(A scalar has zero dimensions)}
Creating multidimensional arrays

- **Dyadic use of $ : reshape**
  
  Generator: i.
  
  i. 10

  0 1 2 3 4 5 6 7 8 9

  2 3 $ i.6

  0 1 2

  3 4 5

  2 3 3 $ i. 4

  NB. 3-d matrix. Displayed in 2-d slices

  0 1 2

  3 0 1

  2 3 0

  1 2 3

  0 1 2

  3 0 1
Indexing (from)

- Dyadic \{ : x \{ v \text{ selects component of } v \text{ at position } x \text{ mod } v \}

\[ 2 \ 0 \ _1 \ _3 \{ \text{“abcdefg”} \]

cage

Result has the shape of the left operand

Negative indices wrap around (index from the right)
Adverbs

• An adverb modifies a verb:
  / distributes a binary operator over a composite argument

  \[ v = .1 \ 2 \ 3 \ 4 \ 5 \]

  \[ + / v \]

  \[ 15 \]

  \[ * / v \]

  \[ 120 \]

  \[ >. / v \]

  \[ 5 \]
Scalar product

The adverb insert and pointwise extension of operators yield the usual operation of linear algebra:

\[ +/- a \ast b \equiv a_0.b_0 + a_1.b_1 + \ldots \]

This generalizes to any dyadic operation:

\[ \text{op}_1 / a \ \text{op} \ b \equiv (a_0 \ \text{op} \ b_0) \ \text{op}_1 (a_1 \ \text{op} \ b_1) \ldots \]
Outer product

Dyadic meaning of adverb insert:
\[ a = . \ 1 \ 2 \ 3 \]
\[ a */ a \]

1 2 3
2 4 6
3 6 9

Generalization: if \( a = 1 \) and \( b = 1 \) then \( a \ op / b \)

Is an object \( R \) such that

\[ R = {a, b} \]

and the \((i, j)\) component of \( R \) is computed as \((i \ {a} \ op \ (j \ {b}))\)
Currying

An argument of a dyad can be bound to provide a monad:

\[ b =: i. 7 \]
\[ b ^ 2 \]
\[ 0 1 4 9 16 25 36 \]

sequence
Squares

\[ \text{Square} =: ^ \& 2 \]
\[ \text{Square} \ b \]
\[ 0 1 4 9 16 25 36 \]

second operand = 2
monadic operation