```
Dyalog APL/W Conference 2011 Unicode Edition
Serial No : 000000
Mon Feb 20 20:24:29 2012
clear ws

      )load wg28
c:\demos\wg28 saved Mon Feb 20 15:19:54 2012

      ⍝ John Scholes - Dyalog APL

      10 × 2 3 4 ⍴ ⍳ 24   ⍝ This is a live APL session.
  0  10  20  30
 40  50  60  70
 80  90 100 110

120 130 140 150
160 170 180 190
200 210 220 230

⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝
⍝ Array: rectanglar collection of items arranged along 0 or more axes,
⍝ where an item is a number, a character or an array.
⍝ Everything is an array. There is no access to underlying scalars!
⍝ NB: APL recycles the word "scalar" to mean rank-0 array.
⍝
⍝ Functions: infix, associate RIGHT with equal precedence.
⍝ Functions take array "arguments" and return array results.
⍝ A "monadic" function takes only a right argument.
⍝
⍝ Operators: infix, associate LEFT with equal precedence.
⍝ Operators curry array or function "operands" to form functions.
⍝ A monadic operator takes only a left operand.
⍝
⍝ Operator-operand binding is stronger than function-argument binding.
⍝ APL has only a single level of high-order function.
⍝
⍝ There is a rich set of primitive functions and operators.
⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝⍝

      42[2 3⍴⊂⍬]      ⍝ indexing a rank-0 array
42 42 42
42 42 42

      2×3+4           ⍝ right-associative, equal precedence
14
```

```
      2+÷-4           ⍝ "monadic" use of functions
1.75
      ⍝ Function definition: {⍺ ∇ ⍵}

        {⌊⍵+0.5} 3.4 ¯5.6         ⍝ round to nearest
3 ¯6
      3 {⍵*÷⍺} 64 125             ⍝ ⍺th root
4 5
      30 {⍵=0:⍵|⍺ ⋄ ⍵ ∇ ⍵|⍺} 105   ⍝ GCD (Euclid)
15
      ⍝ Operator definition: {⍺ (⍺⍺ ∇∇ ⍵⍵) ⍵}
      ⍝                          └────∇────┘

      ÷{⍺⍺ ⍺⍺ ⍵} 4                ⍝ twice (monadic)
4
      0∘≠{⍺⍺ ⍵:⍵⍵ ⍵ ⋄ 0}÷ 4       ⍝ sequential test  (cf C's &&)
0.25
      ⍝ We name things using ←   (right-pointing finger :-)

      vec ← 1 2 3         ⍝ naming an array
      vec = 3 2 1         ⍝ cf: item-wise comparison
0 1 0
      sum ← +/            ⍝ naming a "derived" (curried) function.
      avg ← {(+/⍵)÷⍴⍵}    ⍝ naming a "defined" function ("D-fn").

      sum avg¨ vec(4 5)   ⍝ sum of avg mapped over a 2-vector.
 6.5
      ⍝ A number is a number is a number ...

      ⍳ 5                 ⍝ first five nats
0 1 2 3 4

      ⍳ 6+*∘¯1*÷2         ⍝ ditto - (c) Euler
0 1 2 3 4

      1=49×÷49            ⍝ tolerant comparison
1
      ⍝ Many functions are shape-invariant:

      )copy dfns.dws easter
c:\Home\dfns saved Thu Jan 26 09:52:17 2012
```

```
     ⎕cr'easter'       ⍝ display source of easter function:
easter←{                     ⍝ Easter Sunday in year ⍵.
    G←1+19|⍵               ⍝ year golden number in 19-year Metonic cycle.
    C←1+⌊⍵÷100               ⍝ Century: for example 1984 → 20th century.

    X←¯12+⌊C×3÷4             ⍝ number of years in which leap yr omitted.
    Z←¯5+⌊(5+8×C)÷25         ⍝ synchronises Easter with moon's orbit.

    S←(⌊(5×⍵)÷4)-X+10        ⍝ find Sunday.
    E←30|(11×G)+20+Z-X       ⍝ Epact.
    F←E+(E=24)∨(E=25)∧G>11   ⍝   (when full moon occurs).

    N←(30×F>23)+44-F         ⍝ find full moon.
    N←N+7-7|S+N              ⍝ advance to Sunday.

    M←3+N>31                 ⍝ month: March or April.
    D←N-31×N>31              ⍝ day within month.
    ↑10000 100 1+.×⍵ M D     ⍝ yyyymmdd.
}
      easter 2012                          ⍝ Easter this year April 8th
20120408

      ⎕ ← vec ← ⊂[1 2]2000+2 2 3⍴⍳12       ⍝ vector of matrices
 2000 2001 2002  2006 2007 2008
 2003 2004 2005  2009 2010 2011

      easter vec                           ⍝ vector of easter matrices
 20000423 20010415 20020331  20060416 20070408 20080323
 20030420 20040411 20050327  20090412 20100404 20110424

      ⍝ Some more examples:

      pow ← {(⍺⍺⍣⍺)⍵}              ⍝ Primitive operator ⍣ is power

      0 1 ⍬⍴pow¨ 'hello' 'world'   ⍝ conditional application
 hello  dlrow

      f ← (32∘+)∘(×∘1.8)           ⍝ Operator ∘ is compose / curry
      f ¯273.15 ¯40 0 100          ⍝ Fahrenheit from Celsius
¯459.67 ¯40 32 212

      c ← f⍣¯1                     ⍝ ⍣¯1 is inverse
      c ¯459.67 ¯40 32 212         ⍝ Celsius from Fahrenheit
¯273.15 ¯40 0 100
```

```
      1 +∘÷⍣≡ 1                    ⍝ ⍣≡ is fixpoint
1.618033989

      )copy dfns dft
c:\Home\dfns saved Thu Jan 26 09:52:17 2012

      +∘÷⍣≡ dft 3                  ⍝ display function tree
    ⍣
  ┌─┴─┐
  ∘   ≡
┌─┴─┐
+   ÷

      ⍝ NB:This section experimental - won't work in TryAPL.org
      ⍝ Iverson struggled for years to express calculus: f+g
      ⍝ And then, waking from a nap, he realised:
      ⍝   "Forms f+g and f×g were not used in APL
      ⍝    and could be introduced without conflict"
      ⍝
      ⍝   (f g h)⍵ → (  f ⍵)g(  h ⍵)      ⍝ monadic "fork"
      ⍝  ⍺(f g h)⍵ → (⍺ f ⍵)g(⍺ h ⍵)      ⍝ dyadic "fork"

      mean ← +/ ÷ ⍴        ⍝ mean as fork
      mean 1 2 3 4
2.5
      mean dft 1           ⍝ display of function train
 ┌─┼─┐
 / ÷ ⍴
┌┘
+
      (+,-,×,÷)dft 1       ⍝ display of train (forks of forks)
┌─┼───┐
+ , ┌─┼───┐
    - , ┌─┼─┐
        × , ÷
      6(+,-,×,÷)2          ⍝ vector of fns! (comma not special)
8 4 12 3

      ⍝ Forks (cf S) together with primitive functions:
      2⊢3                  ⍝ "right" (cf I)
3
      2⊣3                  ⍝ "left"  (cf K)
2
      ⍝ and the primitive composition operator f∘g,
      ⍝ provide a combinator base for function/array expressions.
```

```
      ⍝ This means APL {...} "lambda" functions may be converted
      ⍝ mechanically into tacit "combinator" form. Cool or what!

      ⍝ Implementation

      ⍝ APL is usually implemented as an interpreter.
      ⍝ Interpretative overhead amortized over items of large array.

      A←0.1×  1e6?1e6          ⍝ million-item 64-bit float vector.
      B←0.1×1⌈1e6?1e6

      )copy dfns cmpx
c:\Home\dfns saved Thu Jan 26 09:52:17 2012

      cmpx'A÷B'      ⍝ timing in seconds (may use multiple cores).
3.4E¯2

      ⍝ Dyalog "compiler" project.
      ⍝ Pre-evaulation of (small) constant expressions.
      ⍝ Elimination of local names.
      ⍝ Byte code.
      ⍝ General parse-tree topiary.
      ⍝ ...

      ⍝ Semantics: Arrays are passed by value.
      ⍝ Implementation: (no they're not)
      ⍝ APLers like to mutate their arrays:

      A[100?1000]←0   ⍝ replace items of A with zeros

      ⍝ Uses ref-counting to defer copy until/if mutation.

      ⍝ More modern array-languages (J) have removed this mutation
      ⍝ syntax in favour of a pure "merge" operator:
      ⍝   new ← old (selection merge) vals
      ⍝ but the performance considerations remain.

      ⍝ Traditionally, APL has (over-) indulged its users
      ⍝ by allowing them to:
      ⍝ - interrupt the evaluation of an expression,
      ⍝ - save the heap ("workspace") to a binary file,
      ⍝ - update the interpreter executable,
      ⍝ - reload the saved workspace,
      ⍝ - resume execution.
```

```
(8↑''),⊂display demo.links    ⍝ What's next?
```

http://TryAPL.org:8080	⍝ Play APL + more links.
http://dfns.dyalog.com	⍝ Lots of sample code.
www.jsoftware.com	⍝ J: Iverson's successor to APL
www.kx.com	⍝ K: Arthur Whitney's .. ..
www.nars2000.org	⍝ open-source APL (Bob Smith)
www.smartarrays.com	⍝ commercial API for APL-style arrays
Dyalog conference Helsingør.DK, Oct 2012	
Minnowbrook.NY Array-language conference, autumn 2013?	
At a future WG2.8 meeting, please consider inviting:	
Roger Hui (J)	Arthur Whitney (K)

```
⍝ That's all folks! Questions?
```