APL: The Greatest Programming Language You Never Heard Of
Ken Iverson
Prime Numbers

$$2 = \neq 0 = (\tau X) \circ .| \tau X \) / \tau X$$
Rho

\[
\begin{array}{ccc}
3 & 3 & \rho & 1 & 9 \\
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 9 & 9 \\
\end{array}
\]
Select

0 1 0 1 0 1 0 1 0 / 1 8
1 3 5 7
Outer Product

3 4 5 °.+ 1 2 3 4
4 5 6 7
5 6 7 8
6 7 8 9
Identity Matrix

\((I_X) \circ. = I_X\)

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]
<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>
Integral Divisibility Matrix

\[
0 = (\mathbf{1} \cdot \mathbf{X}) \circ \cdot | \mathbf{1} \cdot \mathbf{X}
\]

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
\]
Number of Integral Divisors

\[ + \not \equiv 0 = (\tau X) \circ \cdot | \tau X \]

1 2 2 3 2 4 2 4 3 4
Exactly Two Integral Divisors

\[ 2 = +\not\equiv 0 = (\,\,^1\!X) \circ .| \,\,^1\!X \]

\[
\begin{array}{cccccccccccc}
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Prime Numbers

\[(2 = +\neq 0 = (\iota X) \circ \cdot | \iota X) / \iota X\]

2 3 5 7
More APL Examples

- Leap Year Test
  \[ (0 = 400 \mid X) \lor (0 \not= 100 \mid X)^0 = 4 \mid X \]
- Test for Duplicate Elements
  \[ ^/ (X \tau X) = \tau \rho X \]
- Standard Deviation
  \[ (((+/ (X - (+/ X) \div \rho X) \ast 2) \div \rho X) \ast .5 \]
Iota

(define iota
  (lambda (n)
    (letrec
      ((loop
         (lambda (n acc)
           (if (= n 0)
               acc
               (loop (sub1 n)
                 (cons n acc))))))
     (loop n '()))))

> (iota 8)
(1 2 3 4 5 6 7 8)
(define select
  (lambda (pred)
    (lambda (ls0 ls1)
      (map cdr
        (filter
          (lambda (x) (pred (car x)))
          (map cons ls0 ls1))))))

> ((select even?) (iota 8) '(a b c d e f g h))
(b d f h)
> (map cons '(1 2 3) '(a b c))
  ((1 . a) (2 . b) (3 . c))

> (map (lambda (x) (pred (car x))))
  '((1 . a) (2 . b) (3 . c))
  '((#t . a) (#f . b) (#t . c))

> (filter (lambda (x) (pred (car x))))
  '(((1 . a) (2 . b) (3 . c)))
  '((1 . a) (3 . c))

> (map cdr '((1 . a) (3 . c)))
  (a c)
(define tally
    (lambda (pred)
        (lambda (ls)
            (apply +
                (map (lambda (x) (if (pred x) 1 0)) ls))))))

> ((tally even?) (iota 8))
4
Outer Product

(define outer-product
  (lambda (proc)
    (lambda (us vs)
      (map (lambda (u)
               (map (lambda (v) (proc u v)) vs)) us))))

> ((outer-product cons) '(1 2) '(a b))
(((1 . a) (2 . a)) ((1 . b) (2 . b)))
Prime Numbers

(define primes
  (lambda (n)
    (let ((ls (iota n)))
      ((select (lambda (x) (= x 2))
        (map (tally zero?)
          ((outer-product remainder) ls ls)
          ls))))
    )

> (primes 10)
(2 3 5 7)
Prime Numbers Explained

<table>
<thead>
<tr>
<th>iota</th>
<th>outer-product remainder</th>
<th>tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0 0 2 2 2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0 1 0 3 3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0 0 1 0 4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0 1 2 1 0</td>
<td>2</td>
</tr>
</tbody>
</table>

select

iota

1
2
3
4
5
Goldbach's Conjecture

Every even integer greater than 2 can be expressed as the sum of two primes.
Goldbach's Conjecture

(define goldbach
  (lambda (n)
    (let ((primes (primes n)))
      (apply append
        (apply append
          (apply append
            (outer-product
              (lambda (x y)
                (if (= n (+ x y))
                  (list (list x y))
                  '()))
              primes
              primes)))))))

> (goldbach 98)
((19 79) (31 67) (37 61) (61 37) (67 31) (79 19))
Quines
char data[] = {35,105,110,99,108,117,100,101,32, ... };
#include <stdio.h>

main() {
    int i;

    printf("char data[] = {\n");
    for (i = 0; i < sizeof(data); i++) printf("%d," , data[i]);
    printf("}\n\n\n");
    for (i = 0; i < sizeof(data); i++) printf("%c", data[i]);
}
Scheme Quine

((lambda (x) (list x (list 'quote x)))
 '(lambda (x) (list x (list 'quote x)))))
APL Quine

1ϕ22ρ11ρ""1ϕ22ρ11ρ""
Powerset

\[
\begin{array}{c}
\{a,b,c\} \\
\{a,b\} \\
\{a,c\} \\
\{a\} \\
\{b,c\} \\
\{b\} \\
\{c\} \\
\{\} \\
\end{array}
\]
Results which Grow Fast

- Space complexity gives a lower bound on time complexity.
- A result of size $O(2^n)$ cannot be computed in less than $O(2^n)$ time!
- To grow this fast, a recursive function must call itself twice in every step.
Opening an Oyster
Powerset
> (powerset (cdr '(a b c)))
  (((b c) (b) (c) ())

>(map (lambda (x) (cons 'a x))
  (powerset (cdr '(a b c))))
  (((a b c) (a b) (a c) (a))

(define powerset
  (lambda (xs)
    (if (null? xs)
      '()
      (let ((half (powerset (cdr xs))))
        (append (map (lambda (x) (cons (car xs) x))
                     half)
                half)))))
(define make-change
  (lambda (amount coins)
    (let ((ls (powerset coins)))
      (car (select
            (lambda (x) (= x amount))
            (map (lambda (ls) (apply + ls)) ls) ls))))))
Make Change Explained

> (define half (powerset ' (25 10 10 5 5 5 1 1 1)))
> half

((25 10 10 5 5 5 1 1 1)
 (25 10 10 5 5 5 1 1 1)
 (25 10 10 5 5 5 1 1 1)

.
.
.
(1 1)
(1)
(1 1)
(1)
(1)
())
Make Change Explained

>(map (lambda (ls) (apply + ls)) half)
(63 62 62 61 62 61 61 61 60 58 57 ... 1 2 1 1 0)

>(((select (lambda (x) (= x 57)))
  (map (lambda (ls) (apply + ls)) half))
half)
((25 10 10 5 5 1 1)
(25 10 10 5 5 1 1)
(25 10 10 5 5 1 1)
.  
.  
.  
(25 10 10 5 5 1 1))
Permutations of \{a,b,c\}
Permutations

> (permutations '(b c))
((b c) (c b))

> (permutations (delete 'a '(a b c)))
((b c) (c b))

>(map (lambda (p) (cons 'a p))
    (permutations (delete 'a '(a b c))))
((a b c) (a c b))
Results which Grow Even Faster

- Space complexity gives a lower bound on time complexity.
- A result of size $O(n!)$ cannot be computed in less than $O(n!)$ time!
- To grow this fast, a recursive function must call itself $n$ times in step $n$.
- It can only do this by mapping itself across a list of size $n$. 
Permutations

\[
> (\text{map} \ (\lambda x \ (\text{map} \ (\lambda p \ (\text{cons} \ x \ p)) \ (	ext{permutations} \ \text{(delete} \ x \ \text{'}(a \ b \ c)\text{))}))) \\
\text{'}(a \ b \ c) \\
((a \ b \ c) \ (a \ c \ b) \ (b \ a \ c) \ (b \ c \ a) \ (c \ a \ b) \ (c \ b \ a))
\]

\[
> \ (\text{apply append} \ \\
\text{map} \ (\lambda x \ (\text{map} \ (\lambda p \ (\text{cons} \ x \ p)) \ (	ext{permutations} \ \text{(delete} \ x \ \text{'}(a \ b \ c)\text{))}))) \\
\text{'}(a \ b \ c) \\
((a \ b \ c) \ (a \ c \ b) \ (b \ a \ c) \ (b \ c \ a) \ (c \ a \ b) \ (c \ b \ a))
\]
(define permutations
  (lambda (xs)
    (if (null? xs)
        '(()
        (apply append
          (map (lambda (x)
              (map (lambda (p) (cons x p))
                (permutations (delete x xs)))
          xs))
    )
)
MACHINE  ASSEMBLY  PROCEDURAL  OBJECT ORIENTED  FUNCTIONAL