FP IN THE SHELL

FTS
fishtreesugar@gmail.com

Explanation

Explanation

- Ghost in the Shell

Explanation

- Ghost in the Shell
- FP -- Functional Programming (Scheme)

Explanation

- Ghost in the Shell
- FP -- Functional Programming (Scheme)
- SHELL -- Unix Shell Scripting

UNIX & FP

『一个人如果左脚是臭的，那么他右脚就没有理由不臭。同样，连UNIX的精神都不能认同，他没有理由会认同**FP**的精神。』

-- lichray

Unix Philosophy

- Write programs that do one thing and do it well

- Write programs to work together.

- Write programs to handle text streams, because that is a universal interface.

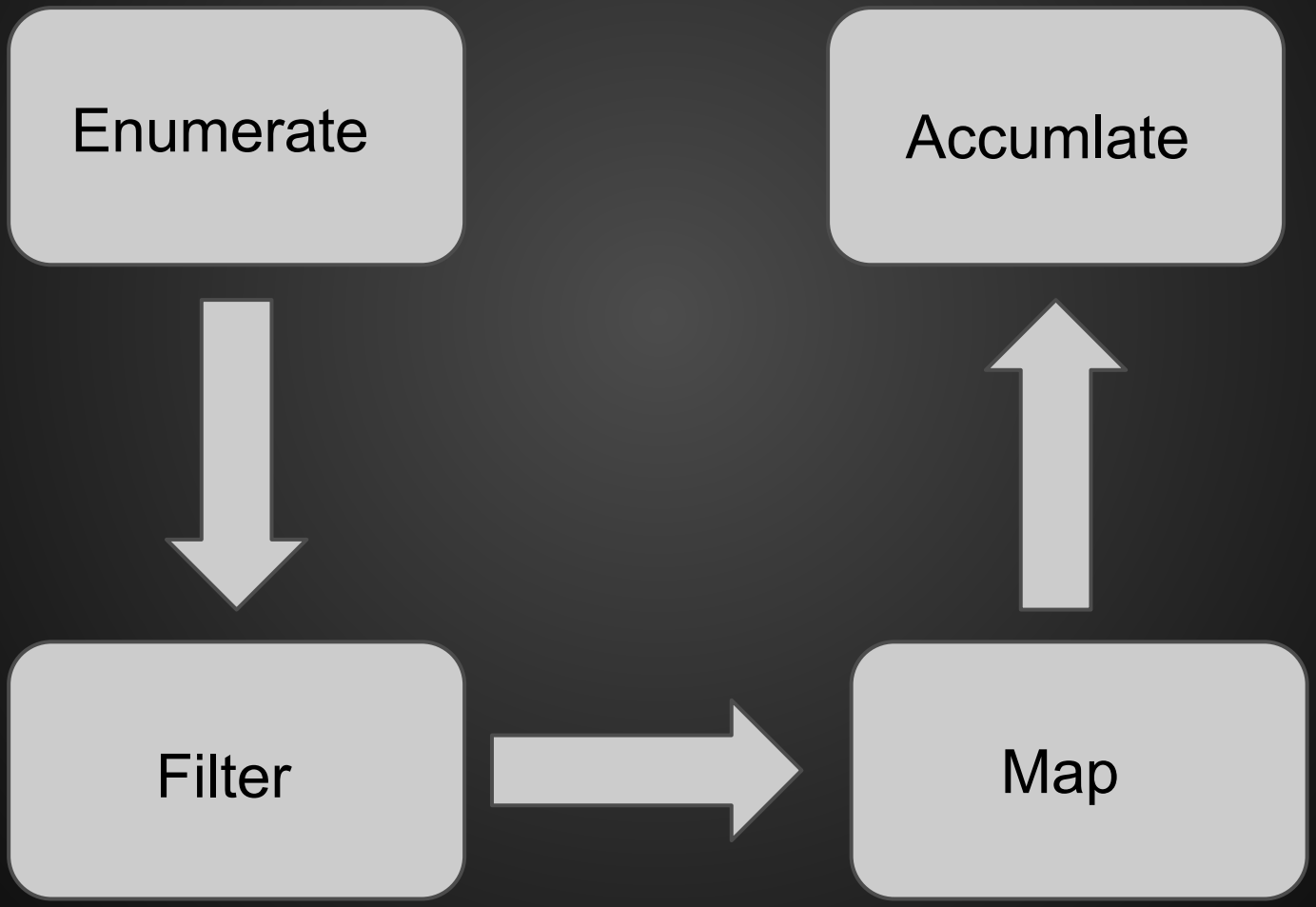
-- Douglas McIlroy

FP Philosophy

- Write function that do one thing and do it well
- Write function to work together.
- Write function to handle list,because that is a universal data structure.

-- FTS

Conventional Interface



Higher-order Function

Higher-order Function

- Dy / Dx

Higher-order Function

- Dy / Dx
- ∫

Higher-order Function

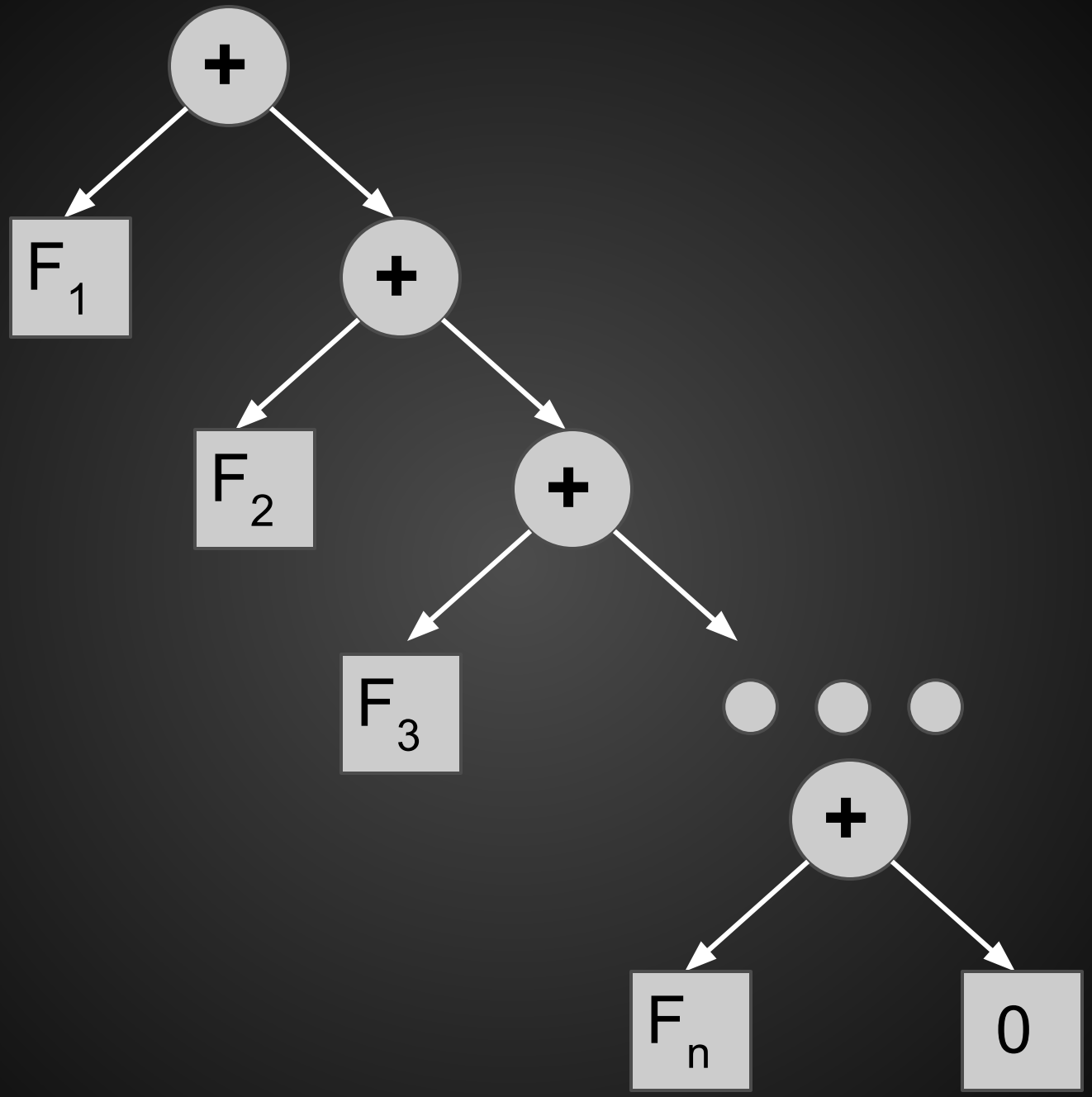
- Dy / Dx
- ∫
- First-class function

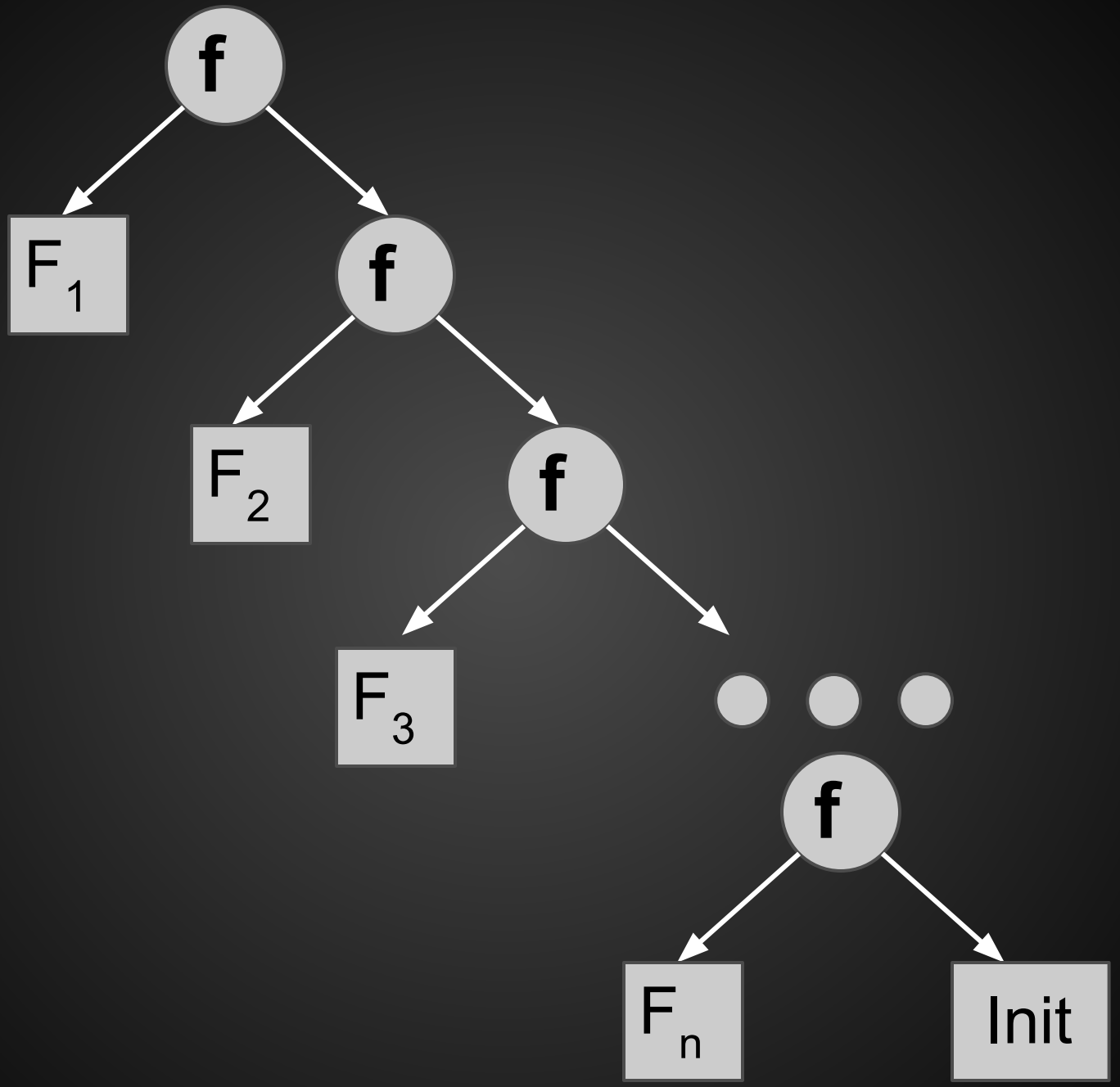
Numerical Derivation

```
(define Δx 0.0001)
(define dy/dx
   (lambda (f)
      (lambda (x) (/ (- (f (+ x Δx))
                        (f x))
                     Δx))))
```

Sigma

```
(define (sigma f low high step)
  (cond ((> low high) 0)
        (else (+ (f low)
                 (sigma f (+ low step)
                        high
                        step)))))
```


+
F1
+
F2
+
F3
+
Fn
0


f
F1
f
F2
f
F3
f
Fn
Init

Accumulator (foldr/reduce/inject)

```
(define (accumulate op initial sequence)
  (if (null? sequence)
      initial
      (op (car sequence)
          (accumulate op initial (cdr sequence)))))

(define (sigma sequence)
          (accumulate + 0 sequence))
```

sequence?

Enumerate

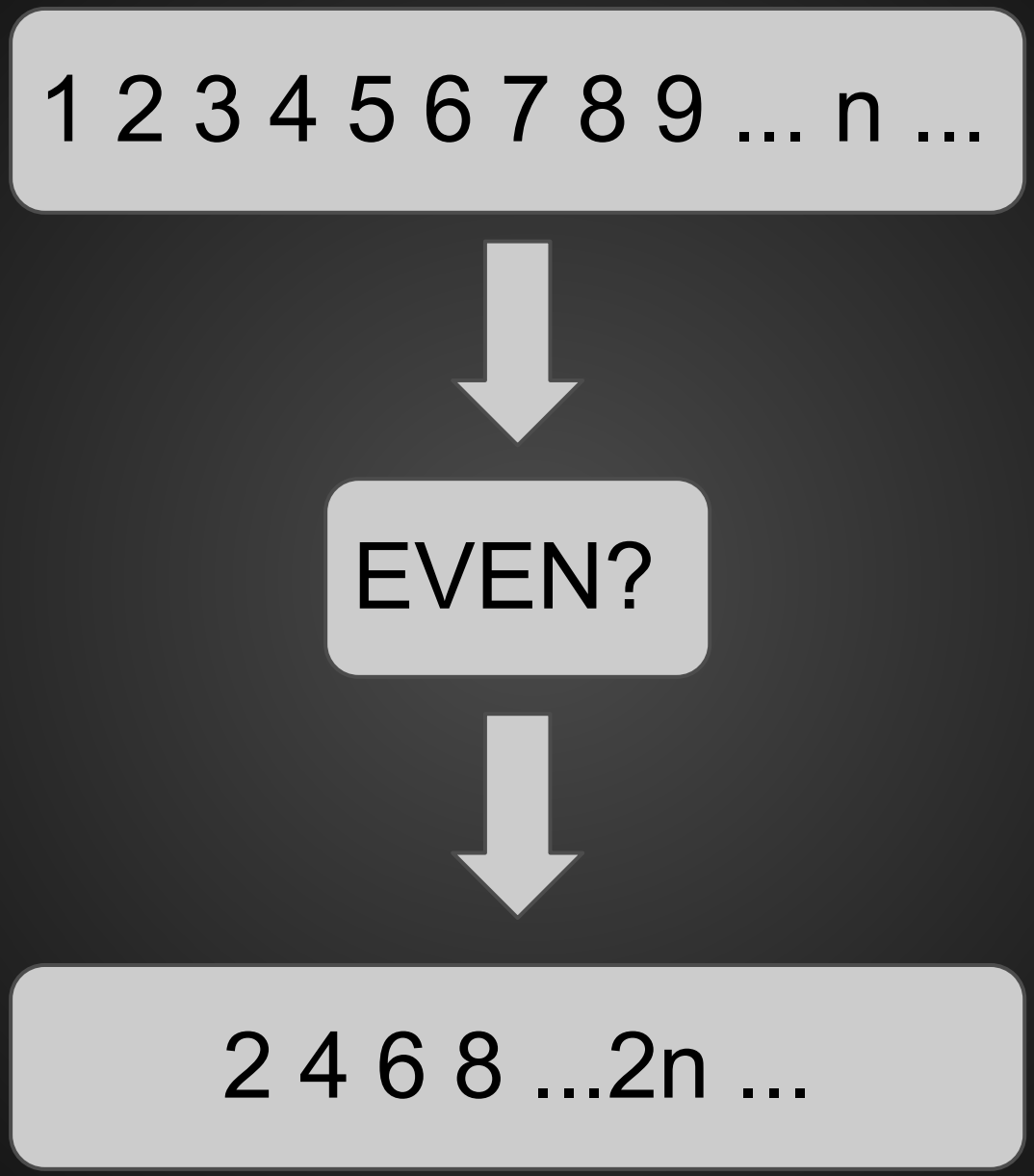
```
(define (enumerate-N low high)
    (if (> low high)
        '()
        (cons low (enumerate-N (+ low 1) high))))
```

enumerate-XXX .....

other way?

Filter

```
(define (filter predicate sequence)
    (cond ((null? sequence) '())
          ((predicate (car sequence))
          (cons (car sequence)
                (filter predicate (cdr sequence))))
          (else (filter predicate (cdr sequence)))))
```


1 2 3 4 5 6 7 8 9 ... n ...
EVEN?
2 4 6 8 ...2n ...

$\sum n^2$?

Map

```
(define (map f sequence)
    (cond ((null? sequence) '())
          (else (cons (f (car sequence))
                      (map f (car sequence))))))
```

Map

```
(define (map f sequence)
    (cond ((null? sequence) '())
              (else (cons (f (car sequence)
                                  (map f (car sequence))))))))
```

```
(define (map f sequence)
    (accumulate
        (lambda (x y) (cons  (f x) y))  '()
        seqeuence))
```

Hash Collisions!?

List comprehension

Quick Sort(Python)

```
def qsort(list):
    if list == []:
        return []
    pivot = list[0]
    l = qsort([x for x in list[1:] if x < pivot])
    u = qsort([x for x in list[1:] if x >= pivot])
    return l + [pivot] + u
```

Quick Sort(Scheme)

```
(define (qsort lst)
  (cond  ((null? lst) '())
    (else (append
            (qsort (filter (lambda (x) (< (car lst) x))
                    (cdr lst)))
            (list (car lst))
            (qsort (filter (lambda (x) (>= (car lst) x)))
                    (cdr lst))))))
```

((((@_@))))

Hello Word?

~~(display "Hello World!")~~

Factorial

Factorial

```
fac(){ seq 1 $1 | (tr '\n' '*' ;echo 1 ) | bc;}
```

Factorial

```
fac(){ seq 1 $1 | (tr '\n' '*' ;echo 1 ) | bc;}
```

```
1'\n'2'\n'3'\n'4'\n'5'\n'6'\n'7'\n'8'\n'9'\n'....$1'\n'
```

Factorial

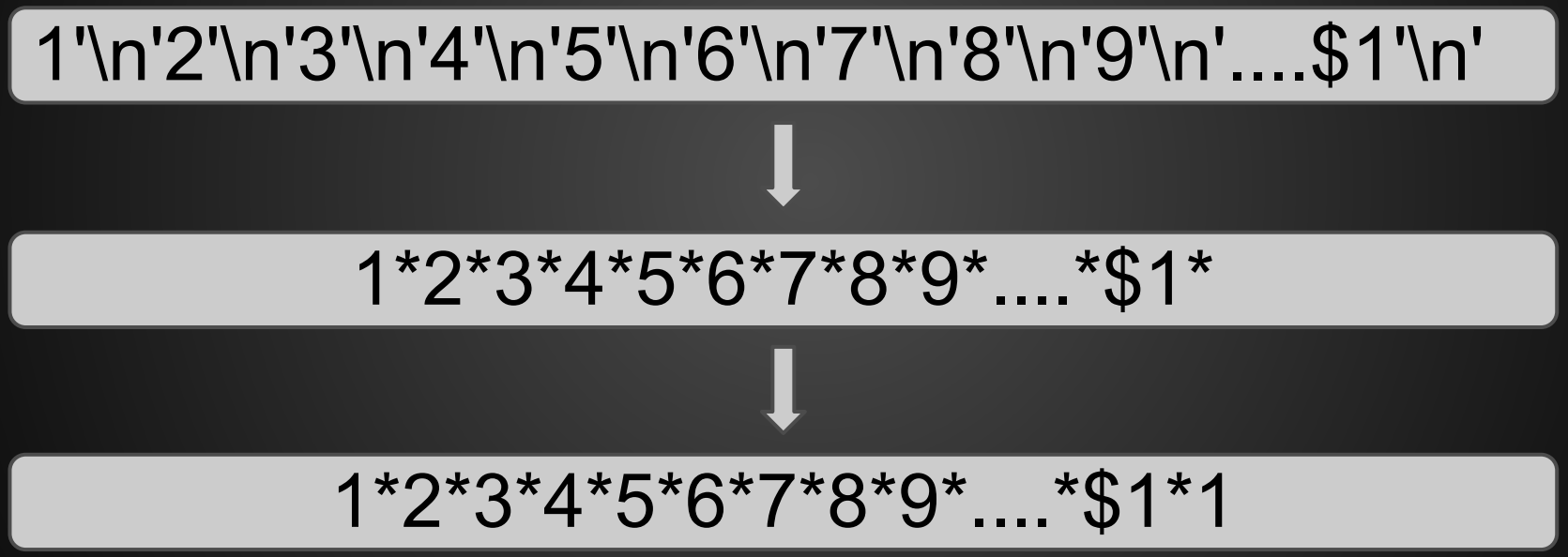
```
fac(){ seq 1 $1 | (tr '\n' '*' ;echo 1 ) | bc;}
```

```
1'\n'2'\n'3'\n'4'\n'5'\n'6'\n'7'\n'8'\n'9'\n'....$1'\n'
```

```
1*2*3*4*5*6*7*8*9*....*$1*
```

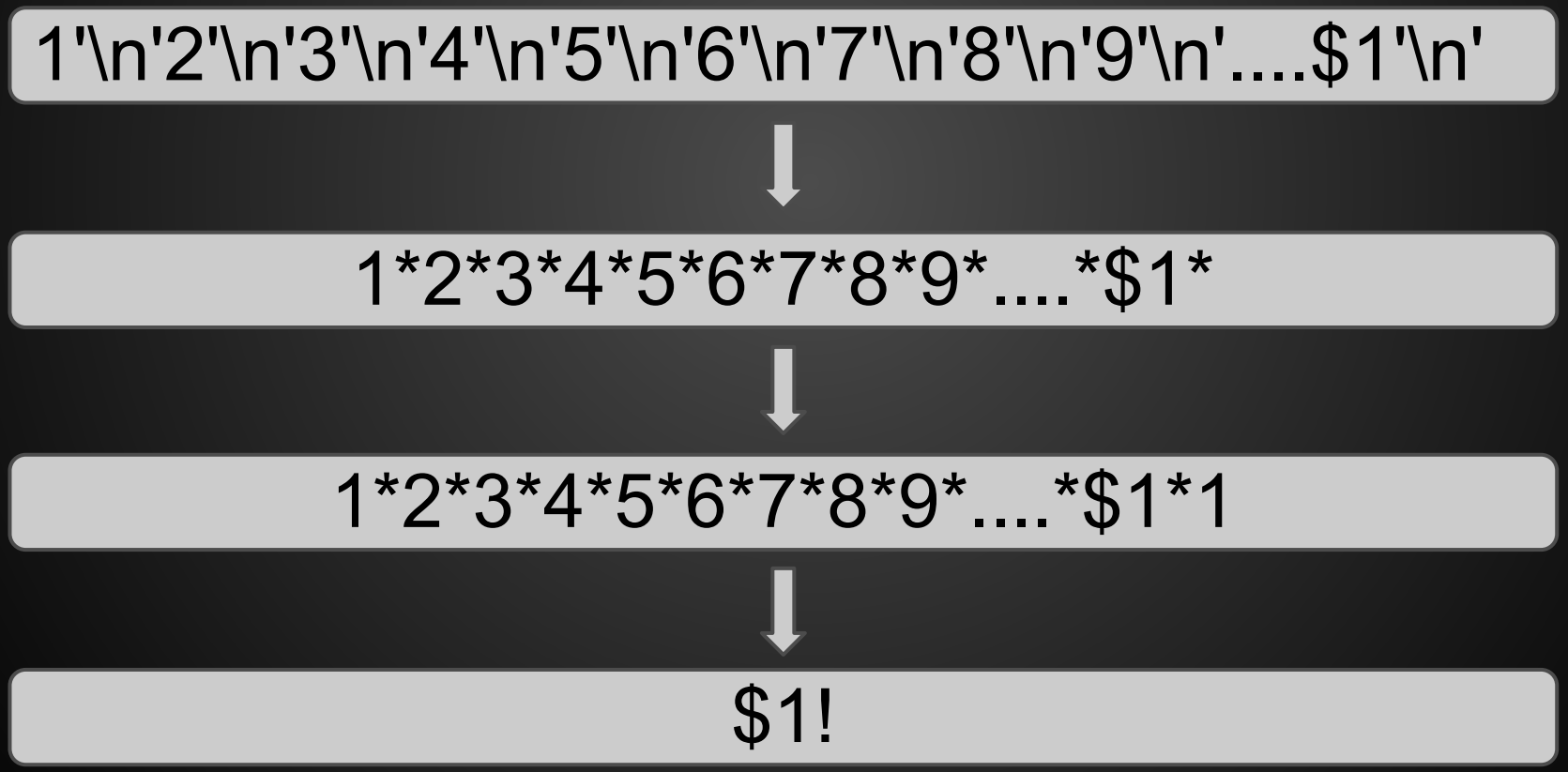
Factorial

```
fac(){ seq 1 $1 | (tr '\n' '*' ;echo 1 ) | bc;}
```



Factorial

```
fac(){ seq 1 $1 | (tr '\n' '*' ;echo 1 ) | bc;}
```



Unix Interface Design Patterns

Unix Interface Design Patterns

- The Source Pattern
  - filter-like program that requires no input

Unix Interface Design Patterns

- The Source Pattern
  - filter-like program that requires no input
- The Filter Pattern
  - Be generous in what you accept, rigorous in what you emit.
  - When filtering, never throw away information you don't need to.
  - When filtering, never add noise

Unix Interface Design Patterns

- The Source Pattern
  - filter-like program that requires no input
- The Filter Pattern
  - Be generous in what you accept, rigorous in what you emit.
  - When filtering, never throw away information you don't need to.
  - When filtering, never add noise
- The Sink Pattern

Enumerlate

- who
- ls
- ps

Yes - most eggache program

Yes - most eggache program

yes
y
y
y
y
y
y
y
y
y

Yes - most eggache program

yes
y
y
y
y
y
y
y
y

yes foo
foo
foo
foo
foo
foo
foo
foo
foo

echo & seq

echo & seq

- echo {a..z}

echo & seq

- echo {a..z}
- echo {1..100} == seq -s' ' 1 10

echo & seq

- echo {a..z}
- echo {1..100} == seq -s' ' 1 10
- echo {01..10} == seq -f"%02g" -s' ' 1 10

echo & seq

- echo {a..z}
- echo {1..100} == seq -s' ' 1 10
- echo {01..10} == seq -f"%02g" -s' ' 1 10
- echo {a,b,c}{1,2,3} & echo {a,b,c}{a,b,c}{a,b,c}

echo & seq

- echo {a..z}
- echo {1..100} == seq -s' ' 1 10
- echo {01..10} == seq -f"%02g" -s' ' 1 10
- echo {a,b,c}{1,2,3} & echo {a,b,c}{a,b,c}{a,b,c}
- echo foo{,,,,,,,,,}

shuf

- echo {1..10} | xargs shuf -e
- shuf -i 1-10

Accumulate

Accumulate

- wc

Accumulate

- wc
- sort/tsort

Accumulate

- wc
- sort/tsort
- bc/dc

Accumulate

- wc
- sort/tsort
- bc/dc
  - seq -f '4/%g' 1 2 99999 | paste -sd-+ | bc -l

Accumulate

- wc
- sort/tsort
- bc/dc
  - seq -f '4/%g' 1 2 99999 | paste -sd-+ | bc -l
- mutt

Accumulate

- wc
- sort/tsort
- bc/dc
  - seq -f '4/%g' 1 2 99999 | paste -sd-+ | bc -l
- mutt
  - echo “IOU” | mutt -s “Dear” -a heart plmm@brain.xx

Filter

The simplest possible filter : cat

The simplest possible filter : cat

The name cat comes from the archaic word “catenate”, which means “to join in a chain”. As all classically educated Unix users know, catena is the Latin word for chain.

-- Harley Hahn's Guide to Unix and Linux

cat - a transformer

cat - a transformer

- combine : cat foo.jpg bar.torrent > baz.jpg

cat - a transformer

- combine : cat foo.jpg bar.torrent > baz.jpg
- create : cat > foo

cat - a transformer

- combine : cat foo.jpg bar.torrent > baz.jpg
- create : cat > foo
- append : cat >> foo

cat - a transformer

- combine : cat foo.jpg bar.torrent > baz.jpg
- create : cat > foo
- append : cat >> foo
- copy : cat < foo > baz

cat - a transformer

- combine : cat foo.jpg bar.torrent > baz.jpg
- create : cat > foo
- append : cat >> foo
- copy : cat < foo > baz
- tac / rev

Set Operations in the Unix Shell

SICP 2.3.3

Set

```
$ cat < set
3
5
1
2
4
```

Set Membership

Set Membership

$ grep -xc 'element' set

Set Membership

```
$ grep -xc 'element' set


# returns 0 if element ∈ set
# returns 1 if element ∉ set
```

Set Equality

Set Equality

```
$ diff -q <(sort -n set1) <(sort -n set2)
```

Set Equality

```
$ diff -q <(sort -n set1) <(sort -n set2)


# returns 0 if set1 = set2
# returns 1 if set1 ≠ set2
```

Set Cardinality

Set Cardinality

```
$ wc -l set | cut -d' ' -f1
```

Set Cardinality

```
$ wc -l set | cut -d' ' -f1

42 foo.c
```

Subset Test

Subset Test

```
$ comm -23 <(sort subset) <(sort set) | head -1
```

Subset Test

```
$ comm -23 <(sort subset) <(sort set) | head -1


# comm returns no output if subset ⊆ set
# comm outputs something if subset ⊊ set
```

Set Union

Set Union

```
$ cat set1 set2 | sort | uniq
```

Set Union

```
$ cat set1 set2 | sort | uniq
```

or

```
$ sort -u set1 set2
```

Set Intersection

Set Intersection

```
$ comm -12 <(sort set1) <(sort set2)
```

Set Intersection

```
$ comm -12 <(sort set1) <(sort set2)
```

or

```
$ sort set1 set2 | uniq -d
```

Set Intersection

```
$ comm -12 <(sort set1) <(sort set2)
```

or

```
$ sort set1 set2 | uniq -d
```

or

```
$ join < (sort  A) < (sort  B)
```

Set Complement

Set Complement

```
$ comm -23 <(sort set1) <(sort set2)
```

Set Complement

```
$ comm -23 <(sort set1) <(sort set2)
```

or

```
$ sort set2 set2 set1 | uniq -u
```

Set Symmetric Difference

```
$ sort set1 set2 | uniq -u
```

Set Symmetric Difference

```
$ sort set1 set2 | uniq -u
```

or

```
comm -3 <(sort set1) <(sort set2) | tr -d '\t'
```

Minimum & Maximum

Minimum & Maximum

min:

```
$ head -1 <(sort set)
```

Minimum & Maximum

min:

```
$ head -1 <(sort set)
```

max:

```
$ tail -1 <(sort set)
```

Set Cartesian Product

Set Cartesian Product

```
$ while read a;
        do while read b;
            do echo "$a, $b";
        done < set1;
   done < set2
```

Evolution

- find
- Sed
- Awk
- Perl

NEXT?

Relational shell programming

Relational shell programming

- cat acts like union
- sed and grep act like selection
- cut acts like projection
- awk can perform renaming
- diff acts (almost) like difference

Reference

- *Structure and Interpretation of Computer Programs* (SICP)
- *The Art of UNIX Programming* (TAOUP)
- *Harley Hahn's Guide to Unix and Linux*
- ** One-Liners Explained*

Thank you