Concepts of Functional Programming

In Search of Purity

by Aaron Feng

You are NOT going to learn a particular language today, just the concepts.

as a potential programmers
they are mentally mutilated
beyond hope of regeneration.

- Dr Edsger W. Dijkstra

Why do we give a sh*t about FP?

Do you care about...

- Testability
- Maintainability
- Scalability
- Reusability

The aliens are here with us
and more are coming

Google's map & reduce algorithm

XMPP with ejabberd

ITA Flight scheduling software

Garbage collection

C#, Javascript, Ruby, Python

Clojure, Scala, F#

Let's go back in time

Lots of smart dudes at Princeton around 1930s

Alonzo Church invented Lambda Calculus in 1932

FP is based on lambda calculus

FP Concepts

Expressions not statements

Higher Order Functions

A function is said to be higher order when it can take other functions as parameters

map(fn, vector)

C# 3.0	*ienum* .Select(*func*)
Ruby	*enum* .collect {*block* } *enum* .map {*block* }
Erlang	lists:map(*Fun* , *List*)
Javascript 1.6	*array* .map(*func*)
Scheme, Clojure	(map *func* *list*)

http://en.wikipedia.org/wiki/Map_%28higher-order_function%29

fold(fn, vector)

C# 3.0	*ienum* .Aggregate(*initval* , *func*)	*ienum* .Reverse().Aggregate (*initval* , *func*)
Ruby	*enum* .inject(*initval*) *&block* *enum* .reduce(*initval*) *&block*	
Erlang	lists:foldl(*Fun* , *Accumulator* , *List*)	lists:foldr(*Fun* , *Accumulator* , *List*)
Javascript 1.8	*array* .reduce(*func* , *initval*)	*array* .reduceRight(*func* , *initval*)
Clojure	(reduce *func initval list*)	

http://en.wikipedia.org/wiki/Fold_%28higher-order_function%29

apply(fn, args)

filter(fn-predicate, vector)

C# 3.0	*ienum* .Where(*pred*)
Ruby	*enum* .find_all {*block* } *enum* .select {*block* }
Erlang	lists:map(*Fun* , *List*)
Javascript 1.6	*array* .filter(*pred*)
Scheme, Clojure	(filter *pred list*)

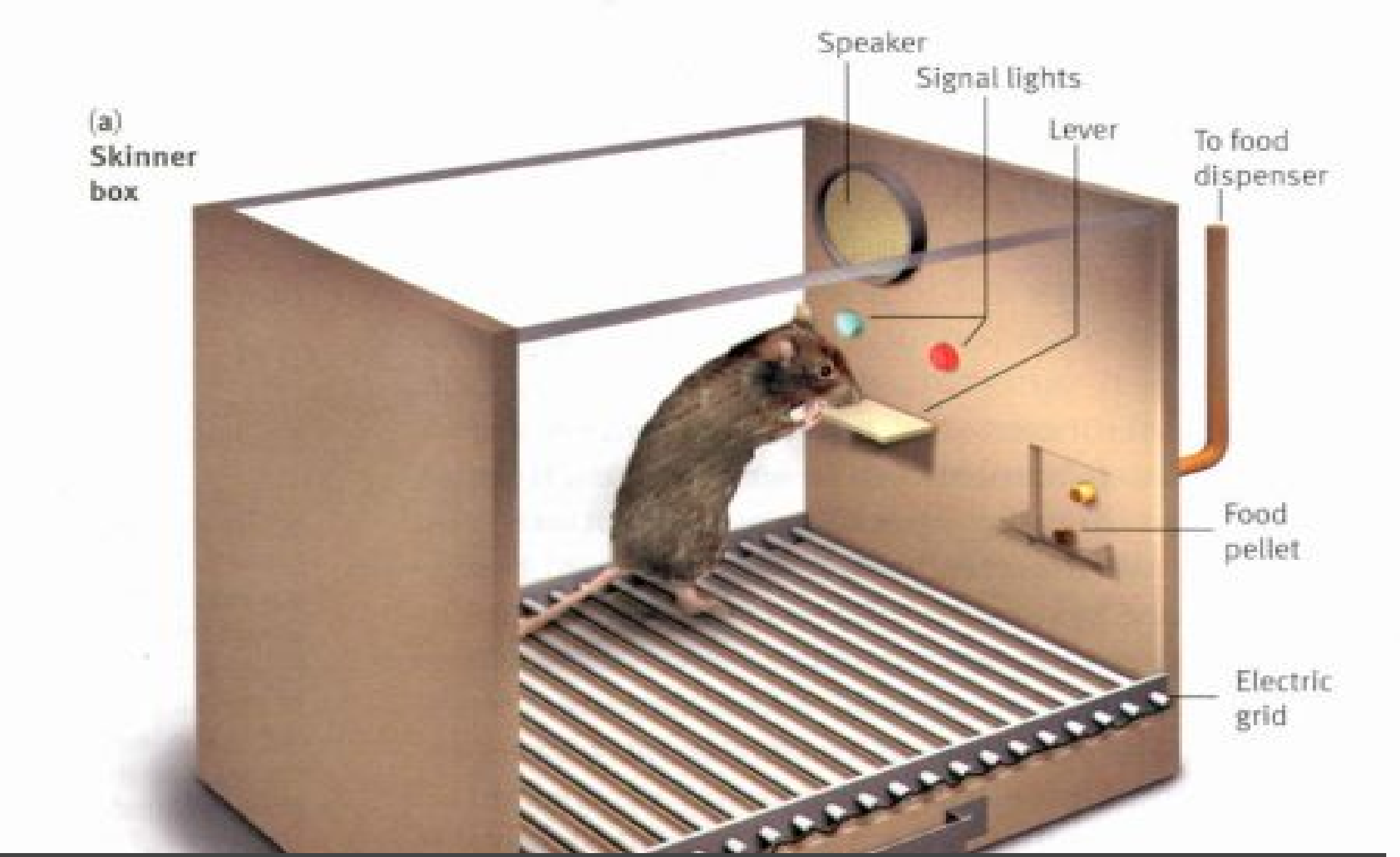
http://en.wikipedia.org/wiki/Filter_%28higher-order_function%29

HOFs is your new abstraction
in your toolbox

Purity

Stop reaching into my pocket!

Immutable (Persistent) data structure



Referential transparency

Non-strict (Lazy) Evaluation

```
v1 = expensiveCalculation1()
v2 = expensiveCalculation2()
magic([v1, v2]) //v1,v2 called?
```

Infinite collection generation

How do you execute the rest of the program?

Control structure

Try to write "if" statement as a function

So why doesn't every language use non-strict evaluation?

IO must be coordinated

Recursion
Stack overflow

```
int factorial(int n) {
  return(n < 2 ? 1 :
    n * factorial(n-1));
}
```

Tail call optimization

```
int factorial_acc(int n, int acc){
  return (n < 2 ? acc :
   factorial_acc(n - 1, n * acc));
}

int factorial(int n){
  return factorial_acc(n, 1);
}
```

Closure

```
bestSellingBooks(threshold) {
   return function() {
       return threshold * magic()
   }
}
```

Currying

```
add(a, b)
inc = add(1)
inc(15) => 16
```

```
// a, b are always the same, make
// them fixed
someFn(a, b, c)
newFn = someFn(1, 2)
newFn(3)
```

So we are back...

- Testability
- Maintainability
- Scalability
- Reusability

Questions?

Examples

```
public delegate void Proc();
public static void Time(Proc fn) {
    var start = DateTime.Now;
    Log("Executing");
    fn();
    Log("Took " + (DateTime.Now -
start).ToString());
  }
```

```
public delegate void Proc<P1>(P1 p);
public static void WithOpen(
  string file, Proc<Stream> fn) {
    using (var stream =
      File.Open(file,FileMode.Open)){
      try {
        fn(stream);
      }
      catch (Exception ex) {
        Log(ex.Message);
      }
  }
}
```

Interesting FP Languages

- Erlang - 1986
- Haskell - 1990
- Lisp - 1958
  - Common Lisp - 1984
  - Scheme - 1975
  - Clojure - 2007
- Scala - 2003
- F# - 2002

Links

- Why Function Programming Matters - http://www.cs.chalmers.se/~rjmh/Papers/whyfp.html
- Functional Programming for the Rest of Us - http://www.defmacro.org/ramblings/fp.html
- Learn you some Erlang - http://learnyousomeerlang.com
- Learn you a Haskell - http://learnyouahaskell.com
- Casting SPELs in Lisp - http://www.lisperati.com/casting.html

Thank you!
twitter: @aaronfeng
email: aaron.feng@gmail.com