Fairy Tale

Clojure

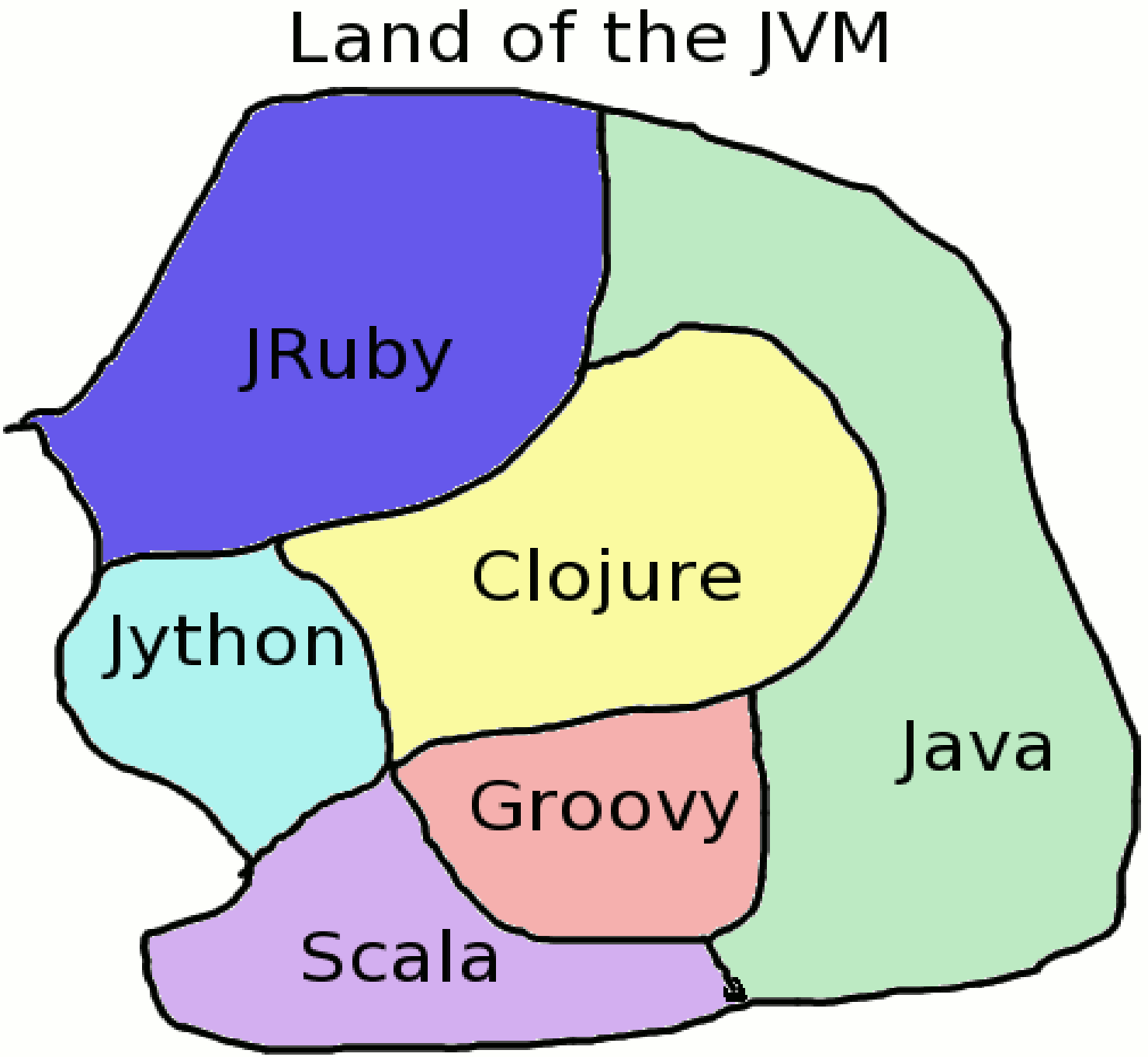
Who am I?

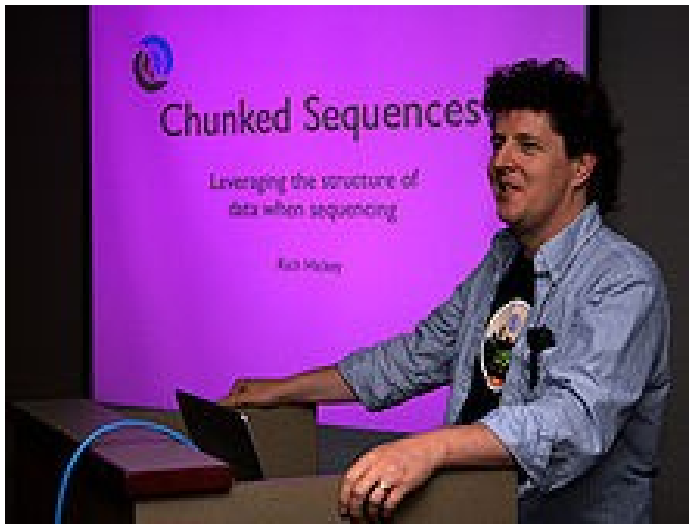
- Work for EdgeCase
- Clojure Enthusiast
- Mother of two young children
- Read lots of Fairy Tales

EdgeCase
SOFTWARE ARTISANS

What about you?

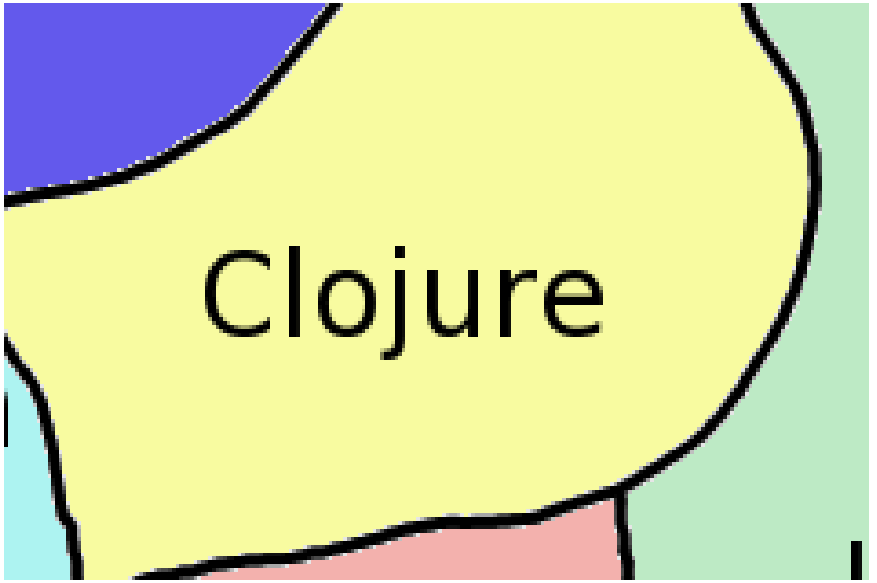
Prologue


Land of the JVM
JRuby
Clojure
Jython
Java
Groovy
Scala



Rich Hickey

Creator of Clojure

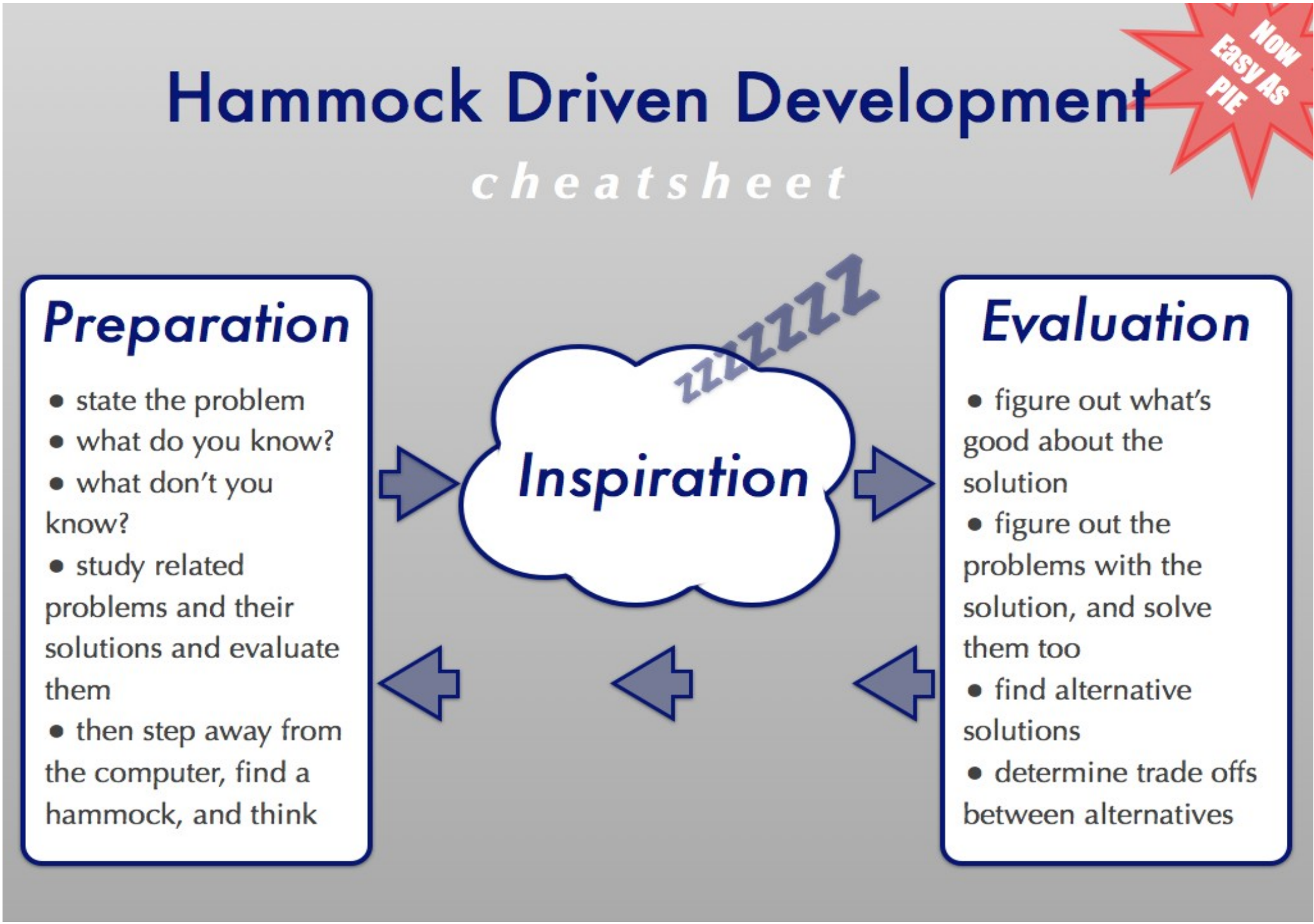


C/ C++

C/ C++

LISP

Hammock Driven Development



From Data Sorcery with Clojure

- First Clojure Public Release Sept 29th 2007
- Clojure 1.0 – May 4th, 2009
- Clojure 1.2 - Aug 19th, 2010
- Clojure 1.3 – Sept 23rd, 2011

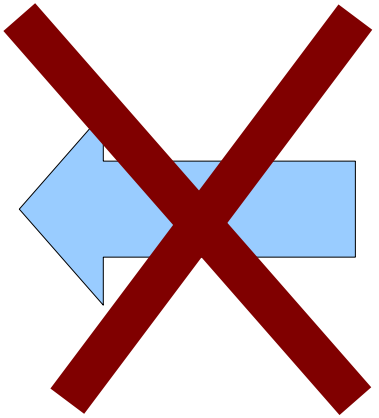
What is Clojure?

What is Clojure?

Dynamic

Dynamic

Princess Crown



Is a Princess Crown

Dynamic

- Expressive
- Concise

  (Lets you focus on the problem)
- Interactive - REPL

What is Clojure?

Dynamic

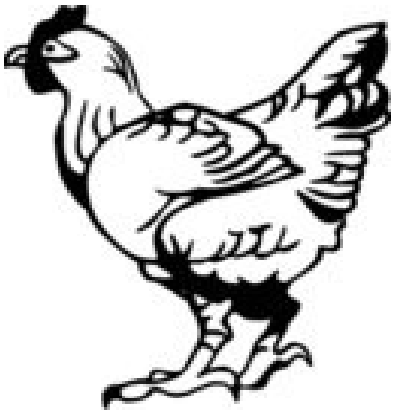
Functional

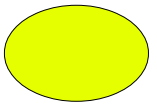
Functional

Functions are “First Class” citizens

Pure Functions – No Side Effects

CLUCK!



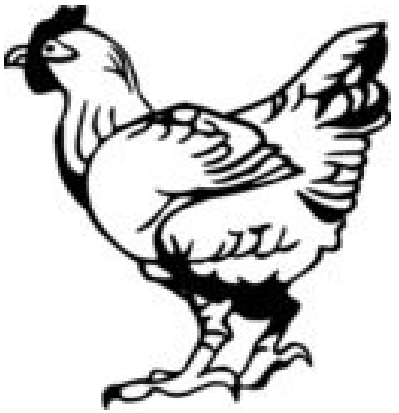


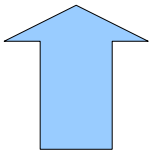
Functional

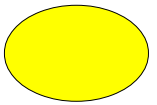
Functions are “First Class” citizens

Pure Functions – No Side Effects

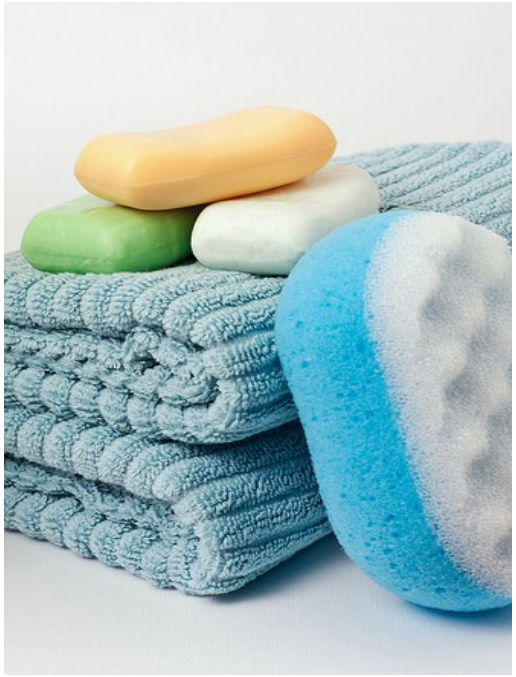
CLUCK!



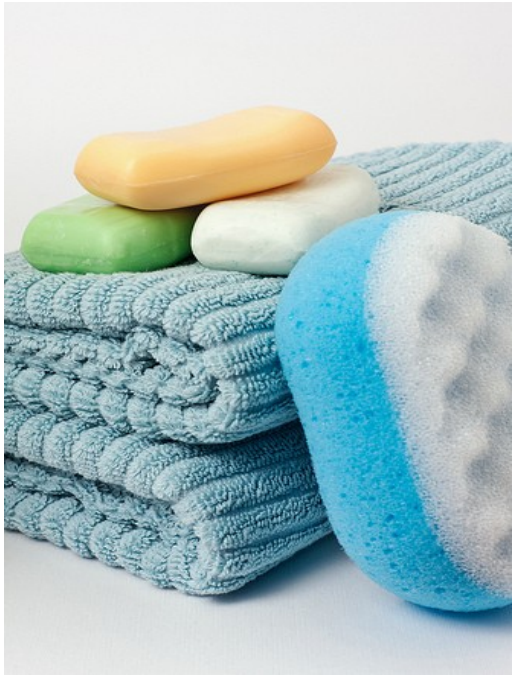




Side Effect



Cleaner Code with Functional Style



Cleaner Code with Functional Style

Object Orientation is Overrated

Born of simulation – now used for everything, even when inappropriate

- Rich Hickey

Mutable stateful objects are the new spaghetti code (Rich Hickey)

Mutable stateful objects are the new spaghetti code (Rich Hickey)

Hard to to understand, test → why we need mock objects and all the unit tests

Concurrency is a disaster

Real life – you need state

Limit, identify and don't scatter it about your code.

Software Transactional Memory (STM)

Approach is similar to a database where updates occur in a transaction. Each thread has its own complete version of the data that it needs.

Clojure:

- Var – mutable reference to object
- Ref - synchronized/ coordinated
- Atom – synchronized/ independent
- Agent – asynchronous/ independent

What is Clojure?

Dynamic

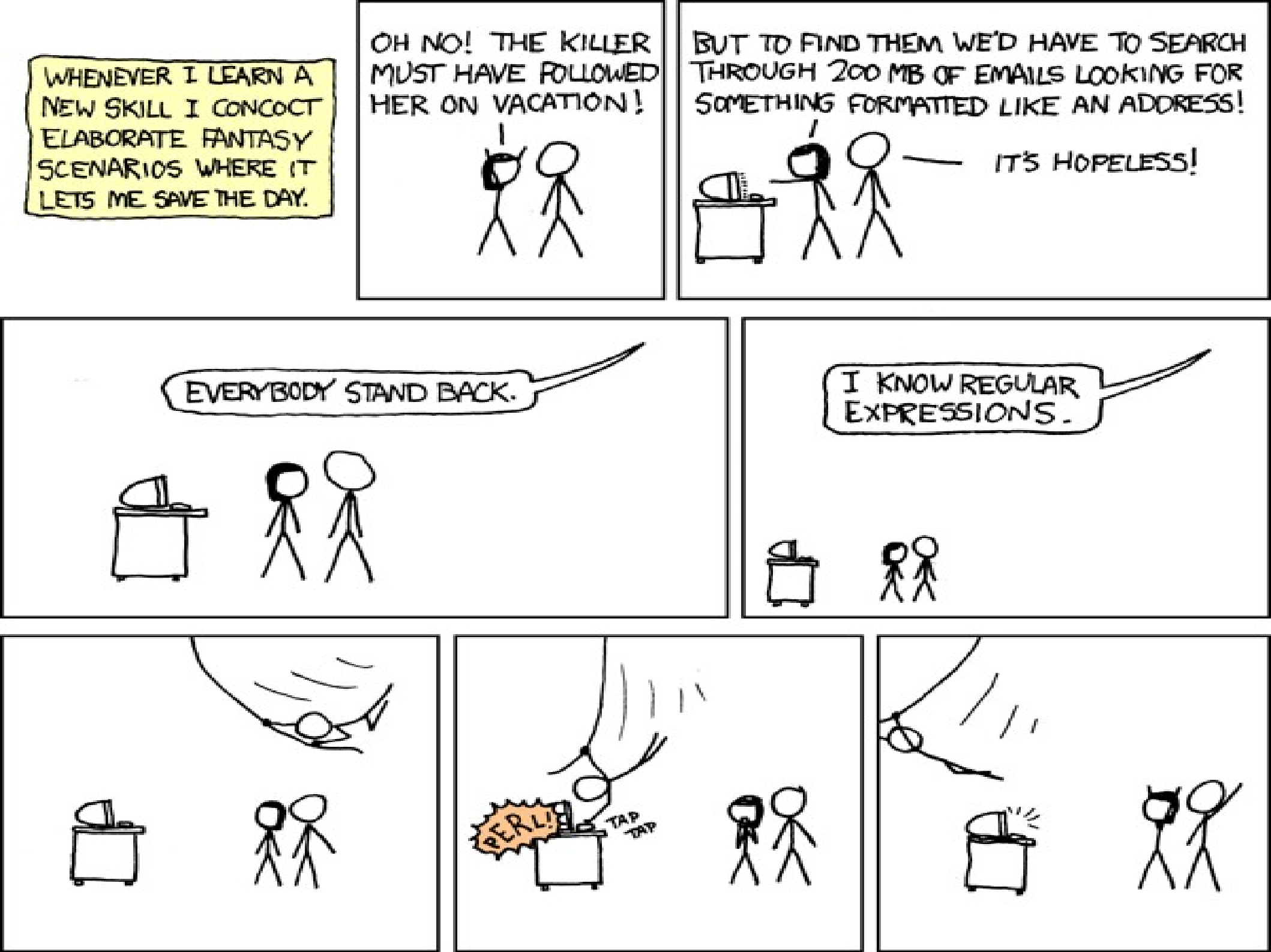
Functional

Concurrent

Immutable Objects + Functional Style = Concurrency

Immutable Objects + Functional Style = Concurrency

Moore's Law Free Lunch is Over.....


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Yeah.
I feel that way about Clojure.

What is Clojure?

Dynamic

Functional

Concurrent

Java Interop

Wrapper Free Access to Java

```
(.toUpperCase "fred")
-> "FRED"
```

Native to the JVM

Solves the New Language Library Problem

What is Clojure?

Macros

Dynamic

Functional

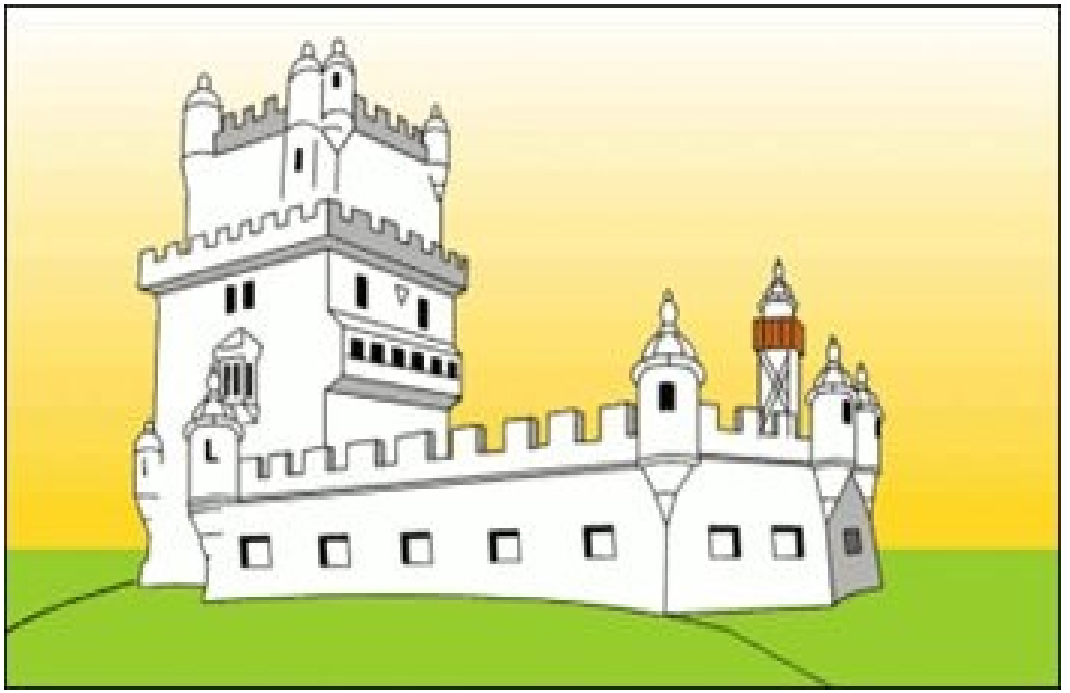
Concurrent

Java Interop

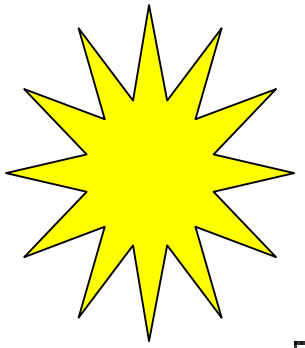
Clojure is a LISP

- Code as Data
- Macros allow you to extend the language
- Macros are executed at compiler pre-process time rather than run-time

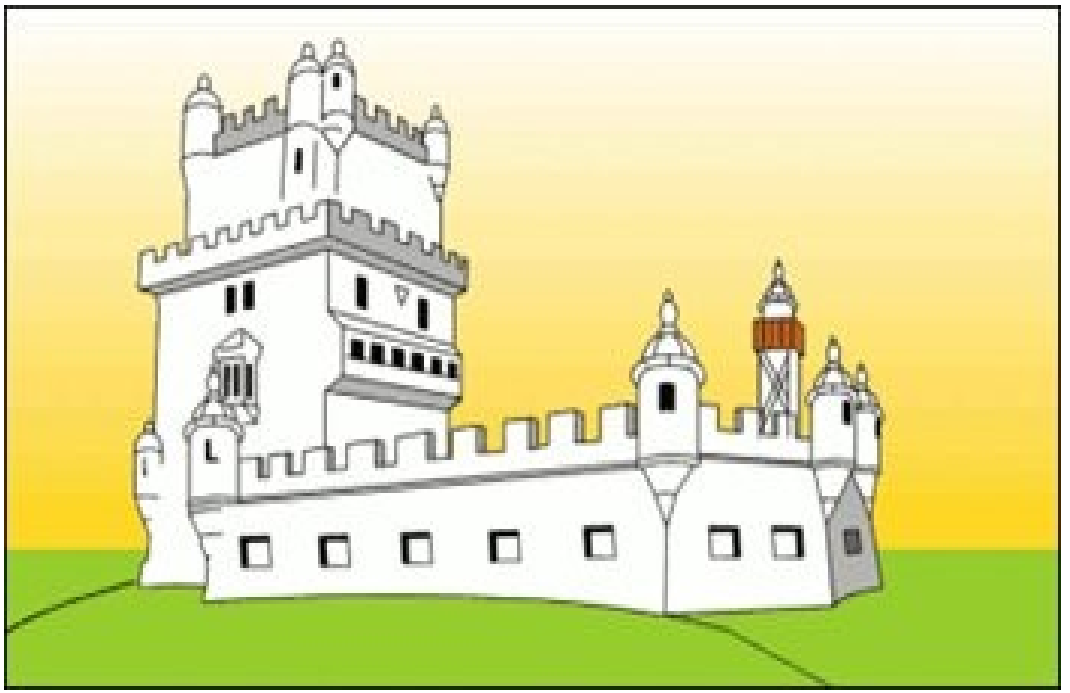
Once upon a time ...



Once upon a time ...



Interactive!
http://tryclj.com/



The Kingdom of Clojureland was ruled by a very good king and queen. It was a very special day. A baby had just been born! The new princess!

user> "Chloe"
"Chloe"

```
user> (.getClass "Chloe")
java.lang.String

user> (.toUpperCase "Chloe")
"CHLOE"
```

She grew from a beautiful baby into a cute toddler and said her first word.

```
user> (println "Mama")
Mama
nil
```

Aww how cute! Her first side effect!

She just loved to make lists

```
user=> '("mom" "dad" "milk" "juice")
("mom" "dad" "milk" "juice")
```

Look Mommy – No commas!

Vectors too.

```
user=> [1 2 "Buckle my shoe"]
[1 2 "Buckle my shoe"]
```

Look Mommy – No commas!

When she got older she started making maps of all her favorite things

```
user=> {:drink "juice", :book "Fox in Socks", :toy "Purple Bear"}
{:drink "juice", :book "Fox in Socks", :toy "Purple Bear"}
```

She learned how to make her first Var

```
user=> (def my-name "Chloe")
#'user/my-name

user=> my-name
"Chloe"
```

She got a time-out for her first lie

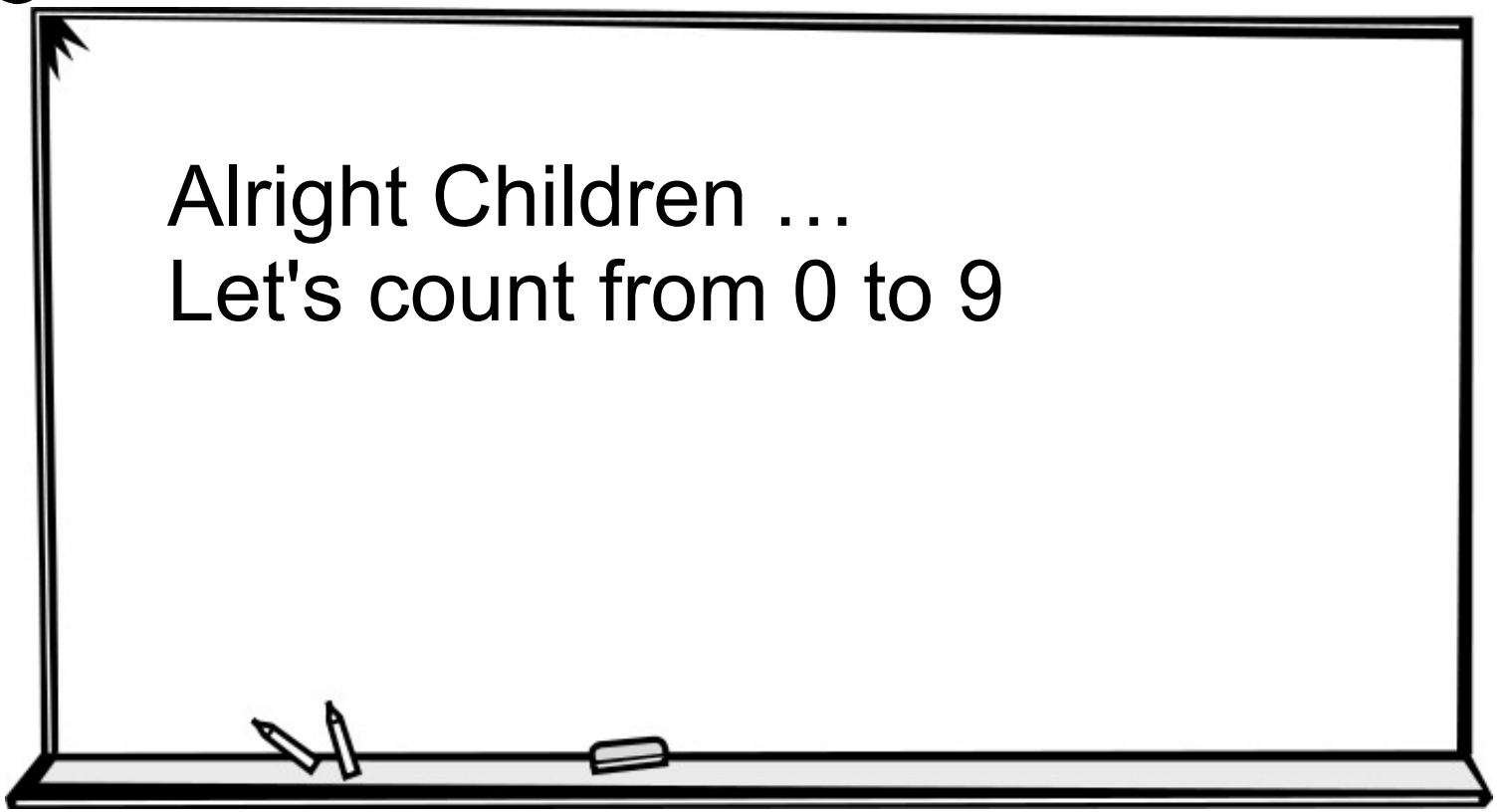
```
user=> (if true "in trouble" "not me")
"in trouble"

user=> (if false "in trouble" "not me")
"not me"

user=> (if nil "in trouble" "not me")
"not me"
```

When she was old enough she got sent to a boarding school for Princesses. She had a hard time fitting in with the other imperative princesses. They made fun of her lisp and said she looked funny with all those parens.

((((((((((()))))))))))

Ha! Ha!

Her teachers gave her a hard time for being different

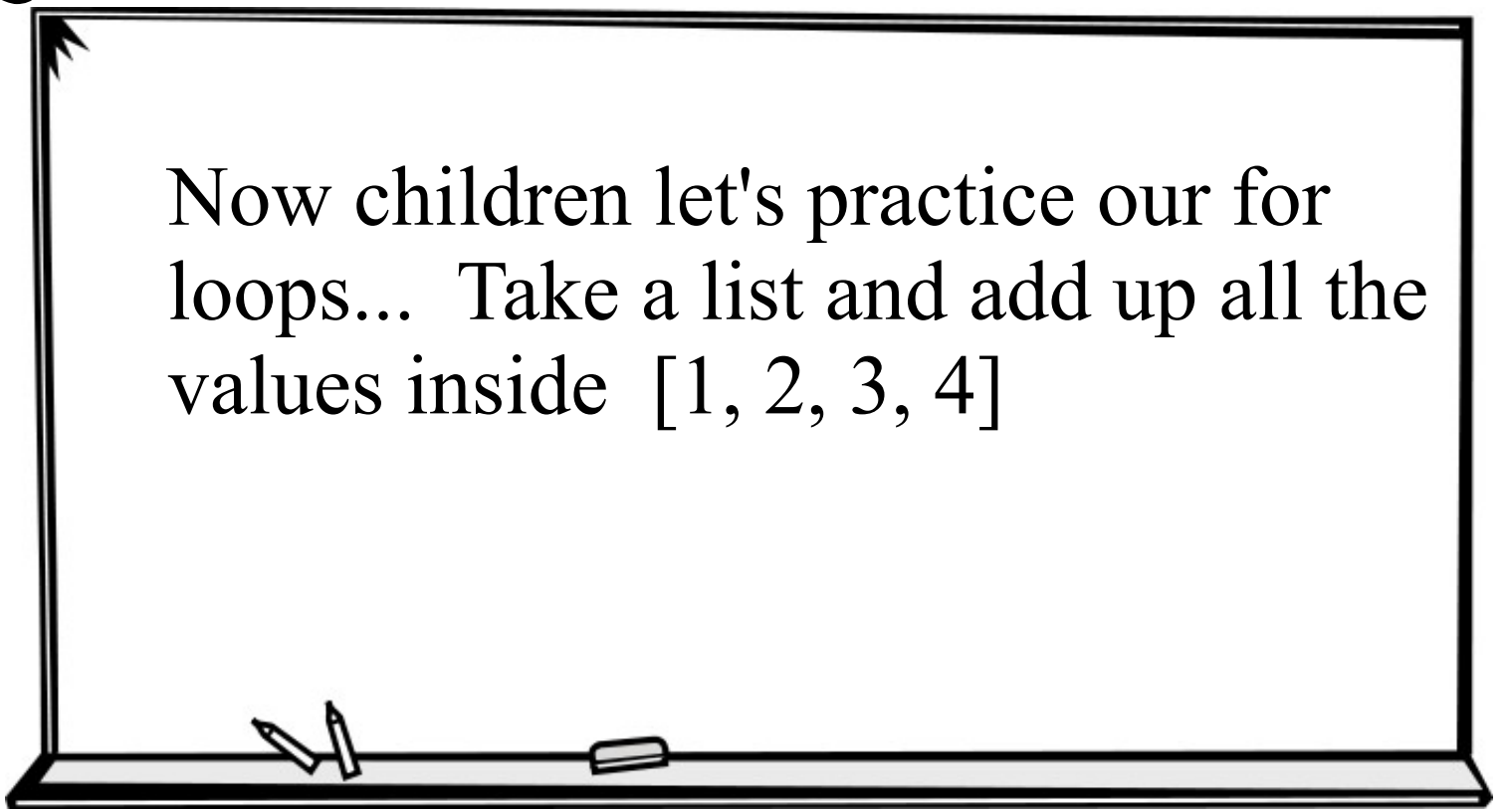



```
user=> (range 10)
(0 1 2 3 4 5 6 7 8 9)
```

Her teacher tried to punish her by having her write "I will not be lazy" 100 times - but it only made it worse when she did:

```
(take 100 (repeat "I will not be lazy"))

("I will not be lazy" "I will not be lazy" "I will not be lazy"
"I will not be lazy" "I will not be lazy" "I will not be lazy" "I
will not be lazy" "I will not be lazy" "I will not be lazy" "I
will not be lazy" "I will not be lazy" "I will not be lazy" "I
```

Her teachers gave her a hard time for being different



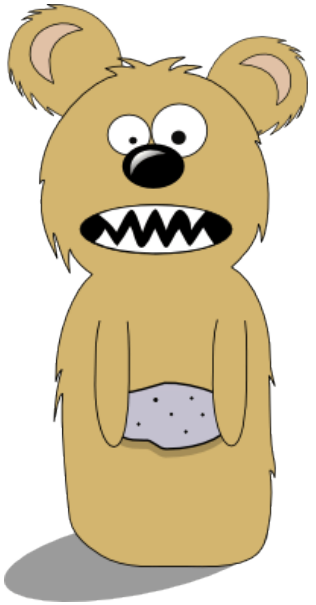

```
user=> (reduce + [1 2 3 4])
10
```

She made it through these hard times with the support and love of her wonderful parents. And finally graduated school and returned home.

```
user=> (defn graduated? [age]
(>= age 18))
#'user/graduated?

user=> (graduated? 18)
true
```

Unfortunately, Tragedy Struck.

Her parents were killed by a freak scheduling accident of the Royal Parade and the Royal Teddy Bear Berzerker's War Games.



The kingdom's evil wizard decided that now was the time to strike...

He would unleash his ultimate monster - The infinite headed hydra

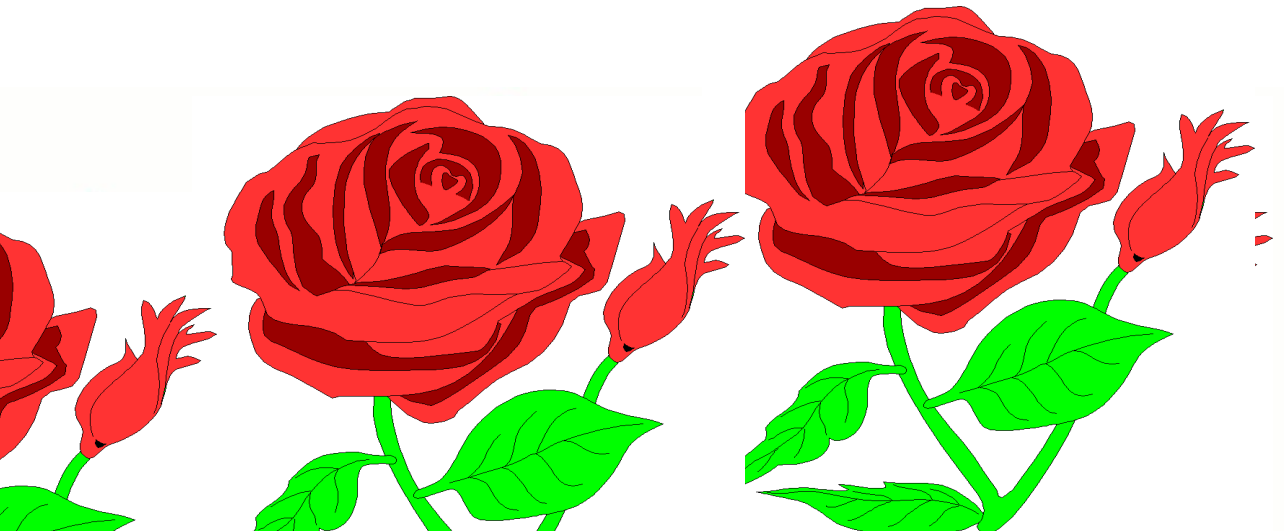
Muha ha!

```
user=> (def hydra (repeat "hydra-head"))
#'user/hydra

user=> (take 5 hydra)
("hydra-head" "hydra-head" "hydra-head" "hydra-head" "hydra-head")
```

The kingdom's Beserker Teddy Bears put up a valiant fight. But the finite troops could not keep up. The Kingdom would be destroyed.

The princess had to do something...

```
user=> (defn defeat-hydra [h]
    (replace {"hydra-head" "flower"} h))
#'user/defeat-hydra

user=> (take 5 (defeat-hydra hydra))
("flower" "flower" "flower" "flower" "flower")

user=> (take 10 (defeat-hydra hydra))
("flower" "flower" "flower" "flower" "flower" "flower"
"flower" "flower" "flower" "flower")
```



After a few rounds - The evil wizard knew he had been beaten and he and the hydra disappeared into the flower scented mists.

Curses Chloe! You've won this round.... but I'll be back!

And everyone had a nice cup of tea.

The End

Epilogue

Who's using Clojure?

FlightCaster

(Clojure Rails) – Clojure for statistical learning

BankSimple

(JRuby Clojure Scala)

TheDeadline

Clojure (Compojure, Ajax)

Want to Learn More?

TryClojure

https://github.com/functional-koans/clojure-koans

4Clojure.com

Github: (gigasquid) yellow-belt-clojure-katas

Books: Programming Clojure
Joy of Clojure
Clojure In Action

Rich Hickey's Ant Colony Demonstration

- World populated with food and ants
- Ants find food, bring home, drop pheromones
- Sense pheromones, food, home
- Ants act independently, on multiple real threads
- Model pheromone evaporation
- Animated GUI
- < 250 lines of Clojure

Credits

Images:

http://www.flickr.com/photos/joeshlabotnik/2054165824/sizes/m/in/photostream/
http://data-sorcery.org/2010/12/29/hammock-driven-dev/
http://www.flickr.com/photos/vikramvetrivel/3912452314/
http://www.flickr.com/photos/gotosira/4699302559/
http://www.flickr.com/photos/horiavarlan/4263957082/
http://www.flickr.com/photos/bobjudge/3444731119/
http://xkcd.com/208/
http://www.flickr.com/photos/26010466@N07/4027386085/sizes/m/in/photostream/
http://www.flickr.com/photos/friarsbalsam/4609212148/sizes/m/in/photostream/
http://www.flickr.com/photos/calliope/159571301/sizes/m/in/photostream/
http://www.flickr.com/photos/seadam/2747922613/