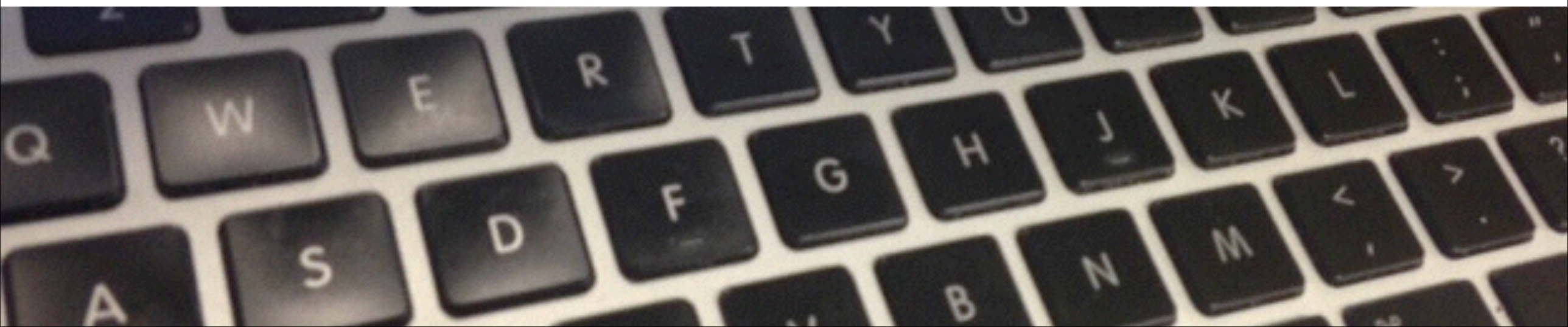
Learning to talk to machines with speech acts

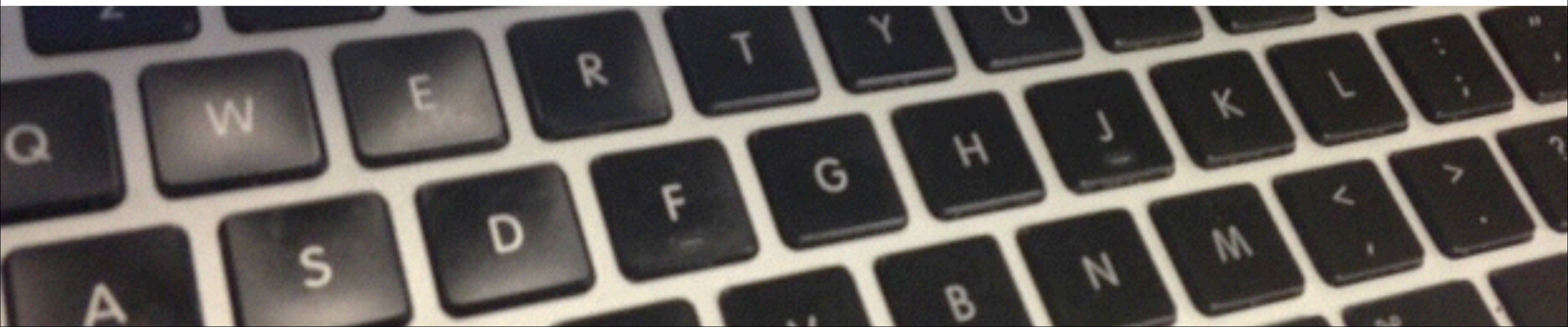
A Shipwrecked Adventure

By @carinmeier

Once Upon A Time


Q
W
E
R
T
Y
A
S
D
F
G
H
J
K
L
B
N
M


CHANCELLOR & SON
DUBLIN


Q
W
E
R
T
Y
A
S
D
F
G
H
J
K
L
B
N
M


STORM CASE
iM2370

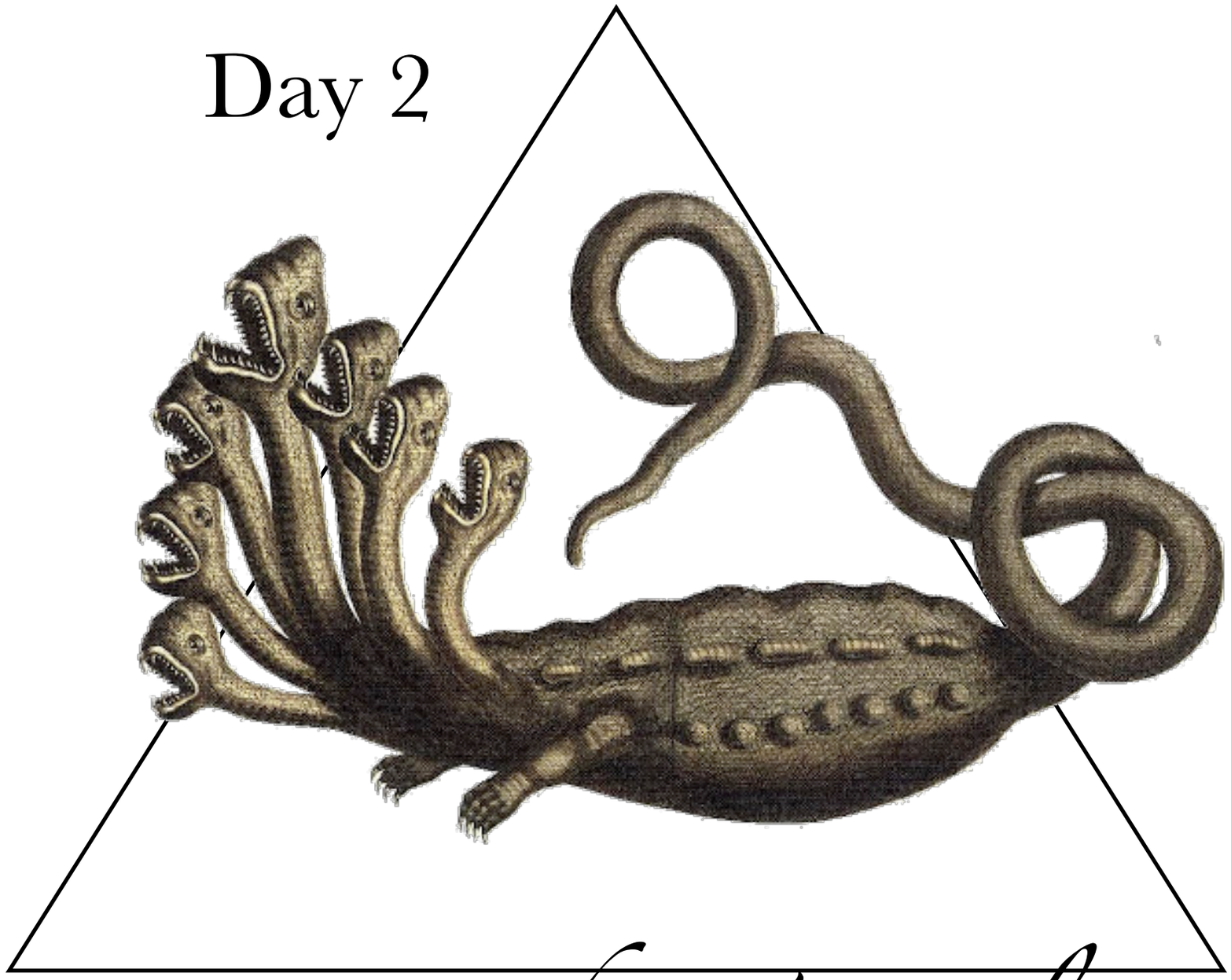
Day 1

Day 2

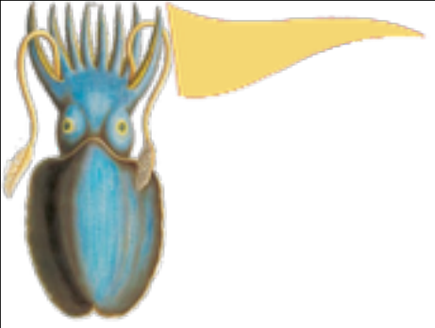
Day 2

Bermuda Triangle

Day 2

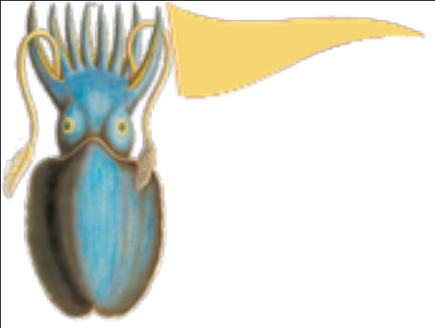


Bermuda Triangle

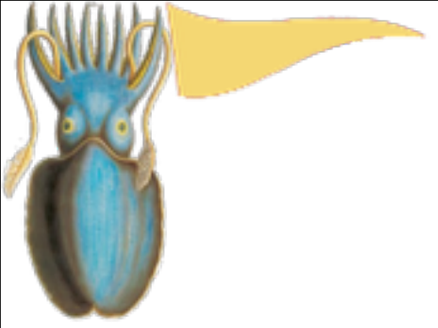


First 24 Hours Critical

First 24 Hours Critical



Ready to Code



How to build a language?

README.md

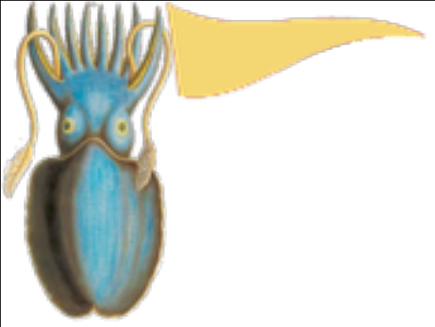
Instaparse 1.2.2

What if context-free grammars were as easy to use as regular expressions?

Features

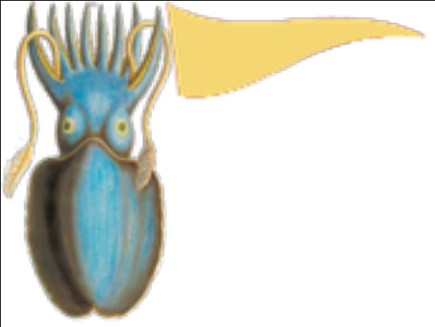
Instaparse aims to be the simplest way to build parsers in Clojure.

- Turns *standard EBNF or ABNF notation* for context-free grammars into an executable parser that takes a string as an input and produces a parse tree for that string.
- *No Grammar Left Behind*: Works for *any* context-free grammar, including *left-recursive*, *right-recursive*, and *ambiguous* grammars.
- Extends the power of context-free grammars with PEG-like syntax for lookahead and negative lookahead.
- Supports both of Clojure's most popular tree formats (hiccup and enlive) as an output target.
- Detailed reporting of parse errors.
- Optionally produces lazy sequence of all parses (especially useful for diagnosing and debugging ambiguous grammars).
- "Total parsing" mode where leftover string is embedded in the parse tree.
- Optional combinator library for building grammars programmatically.
- Performant.



How to build a language?

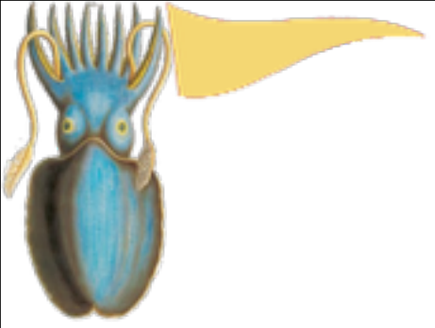
```
MyCoolLang> 1
1
```



How to build a language?

```
(ns coollang.parser
  (:require [instaparse.core :as insta]))
```

```
(def parser
  (insta/parser
   "number = #'[0-9]+'"))

(parser "1") ;=> [:number "1"]
```

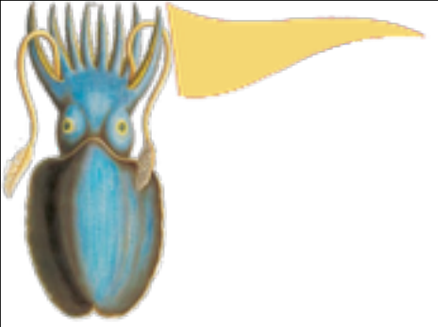


How to build a language?

```
(ns coollang.parser
  (:require [instaparse.core :as insta]))
```

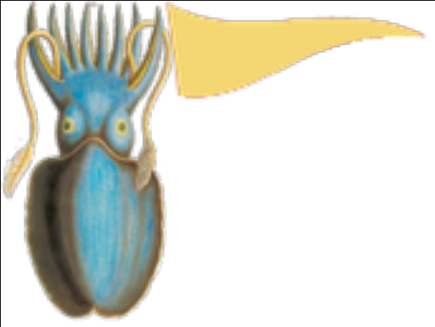
```
(def parser
  (insta/parser
   "number = #'[0-9]+'"))
```

```
(parser "1") ;=> [:number "1"]
```



How to build a language?

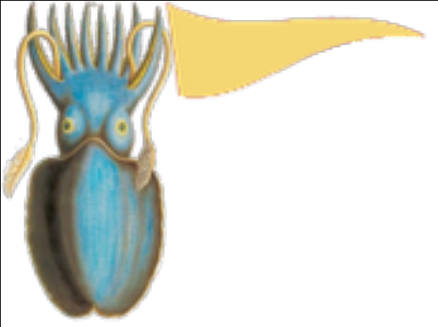
```
(ns coollang.parser
  (:require [instaparse.core :as insta]))

(def parser
  (insta/parser
   "number = #'[0-9]+'"))

(parser "1") ;=> [:number "1"]
```



How to build a language?

```
(def transform-options
  {:number read-string})
```

```
(defn parse [input]
  (->> (parser input)
       (insta/transform transform-options)))

(parse "1") ;=> 1
```

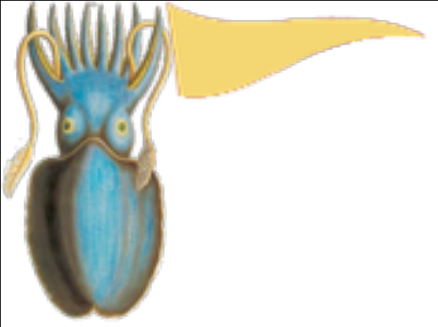


How to build a language?

```
(def transform-options
  {:number read-string})
```

```
(defn parse [input]
  (->> (parser input)
       (insta/transform transform-options)))
```

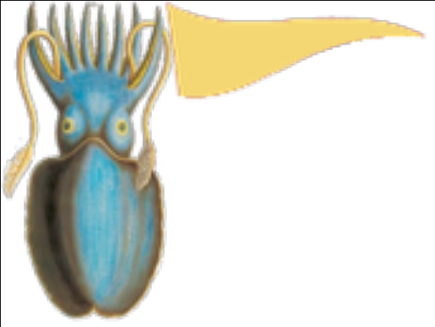
```
(parse "1") ;=> 1
```



How to build a language?

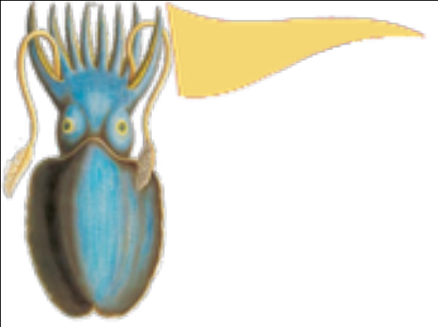
```
(def transform-options
  {:number read-string})

(defn parse [input]
  (->> (parser input)
       (insta/transform transform-options)))
```

```
(parse "1") ;=> 1
```



How to build a language?

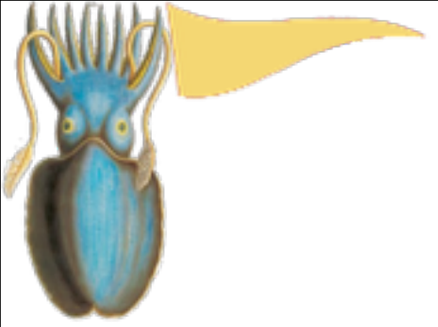
```
MyCoolLang> 1 2 3 4
[1 2 3 4]
```



How to build a language?

```
(def parser
  (insta/parser
   "expr = number | vector
    vector = ((space)* number (space)*)+
    <space> = <#'[\\s\\t\\n\\,]+'>
    number = #'[0-9]+'"))
```

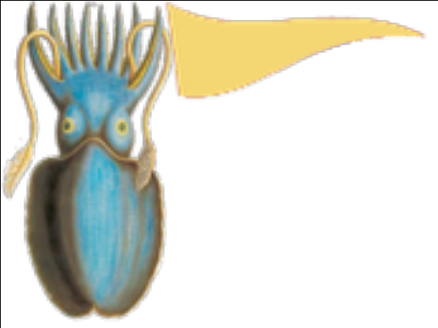
```
(parser "1 2 3 4") ;=>
    ;;[:expr
    ;;  [:vector [:number "1"]
    ;;           [:number "2"]
    ;;           [:number "3"]
    ;;           [:number "4"]]]
```



How to build a language?

```
(def parser
  (insta/parser
   "expr = number | vector
    vector = ((space)* number (space)*)+
    <space> = <#'[\\s\\t\\n\\,]+'>
    number = #'[0-9]+'"))
```

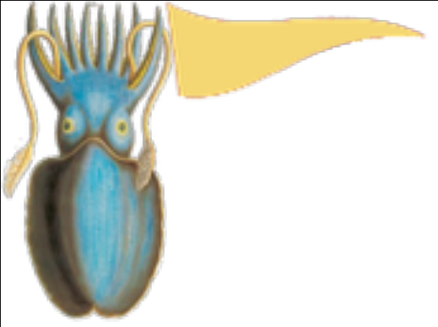
```
(parser "1 2 3 4") ;=>
    ;;[:expr
    ;;  [:vector [:number "1"]
    ;;           [:number "2"]
    ;;           [:number "3"]
    ;;           [:number "4"]]]
```



How to build a language?

```
(def transform-options
  {:number read-string
   :vector (comp vec list)
   :expr identity})
```

```
(defn parse [input]
  (->> (parser input)
       (insta/transform transform-options)))

(parse "1 2 3 4") ;=> [1 2 3 4]
```

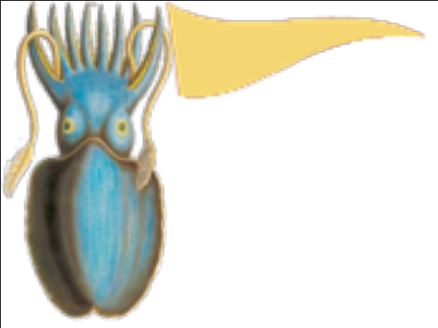


How to build a language?

```
(def transform-options
  {:number read-string
   :vector (comp vec list)
   :expr identity})
```

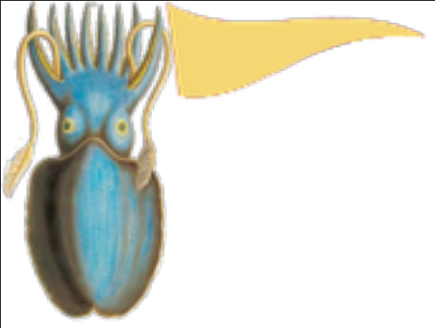
```
(defn parse [input]
  (->> (parser input)
       (insta/transform transform-options)))
```

```
(parse "1 2 3 4") ;=> [1 2 3 4]
```



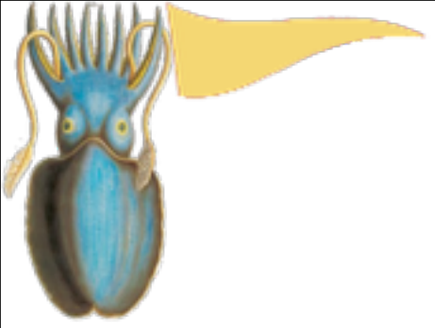
How to build a language?

```
(def transform-options
  {:number read-string
   :vector (comp vec list)
   :expr identity})

(defn parse [input]
  (->> (parser input)
       (insta/transform transform-options)))

(parse "1 2 3 4") ;=> [1 2 3 4]
```



How to build a language?

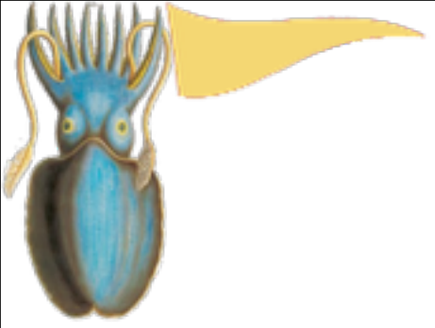
```
MyCoolLang> + 1 2 3 4
10
```



How to build a language?

```
(def parser
  (insta/parser
   "expr = number | vector | operation
    operation = operator space+ vector
    operator = '+' | '-' | '*' | '/'
    vector = ((space)* number (space)*)+
    <space> = <#'[\\s\\t\\n\\,]+'>
    number = #'[0-9]+'"))
```

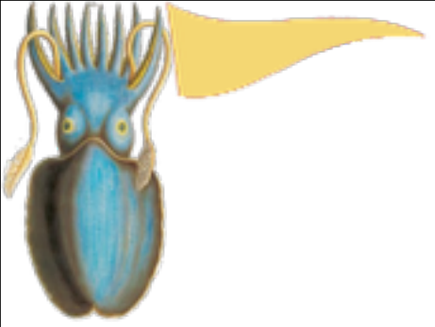
```
(parser "+ 1 2 3 4");=>
;;[:expr
;; [:operation
;;  [:operator "+"]
;;  [:vector [:number "1"] [:number "2"]
;;           [:number "3"] [:number "4"]]]]
```



How to build a language?

```
(def parser
  (insta/parser
   "expr = number | vector | operation
    operation = operator space+ vector
    operator = '+' | '-' | '*' | '/'
    vector = ((space)* number (space)*)+
    <space> = <#'[\\s\\t\\n\\,]+'>
    number = #'[0-9]+'"))
```

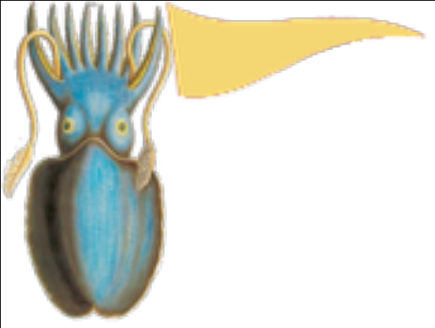
```
(parser "+ 1 2 3 4");=>
;;[:expr
;; [:operation
;;  [:operator "+"]
;;  [:vector [:number "1"] [:number "2"]
;;           [:number "3"] [:number "4"]]]]
```



How to build a language?

```
(defn choose-operator [op]
  (case op
    "+" +
    "-" -))
```

```
(def transform-options
  {:number read-string
   :vector (comp vec list)
   :operator choose-operator
   :operation apply
   :expr identity})

(defn parse [input]
  (->> (parser input)
       (insta/transform transform-options)))

(parse  "+ 1 2 3 4") ;=> 10
```

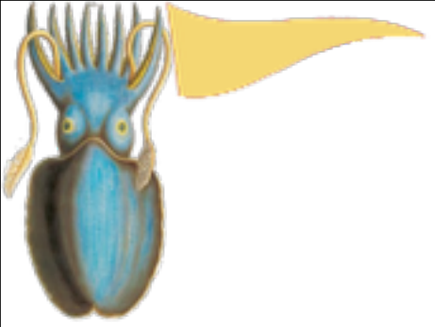


How to build a language?

```
(defn choose-operator [op]
  (case op
    "+" +
    "-" -))
```

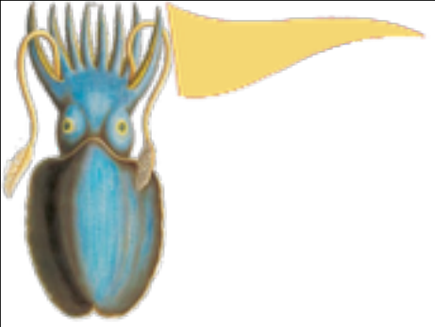
```
(def transform-options
  {:number read-string
   :vector (comp vec list)
   :operator choose-operator
   :operation apply
   :expr identity})
```

```
(defn parse [input]
  (->> (parser input)
       (insta/transform transform-options)))

(parse  "+ 1 2 3 4") ;=> 10
```



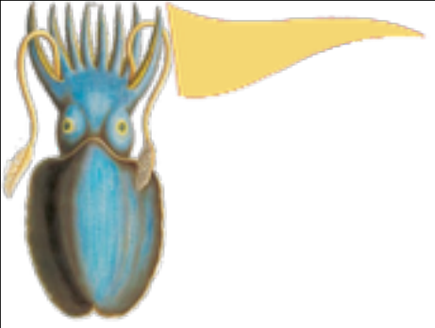
How to build a language?

```
(defn choose-operator [op]
  (case op
    "+" +
    "-" -))

(def transform-options
  {:number read-string
   :vector (comp vec list)
   :operator choose-operator
   :operation apply
   :expr identity})

(defn parse [input]
  (->> (parser input)
       (insta/transform transform-options)))

(parse  "+ 1 2 3 4") ;=> 10
```



How to build a language?

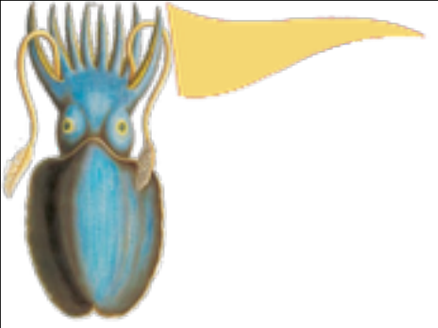
```
(defn choose-operator [op]
  (case op
    "+" +
    "-" -))

(def transform-options
  {:number read-string
   :vector (comp vec list)
   :operator choose-operator
   :operation apply
   :expr identity})

(defn parse [input]
  (->> (parser input)
       (insta/transform transform-options)))

(parse  "+ 1 2 3 4") ;=> 10
```



How to build a language?

What about a REPL?

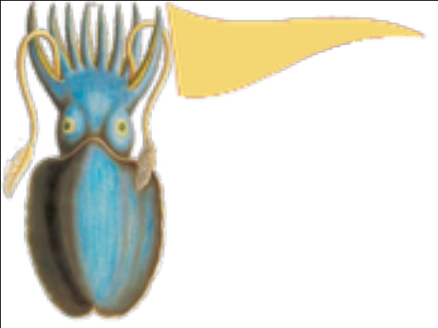
```
MyCoolLang>
```



How to build a language?

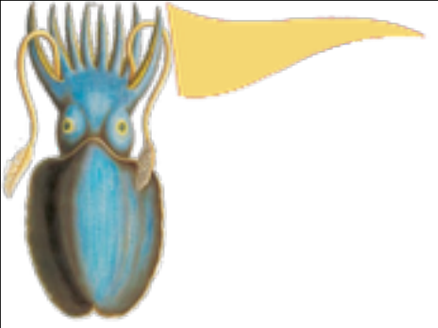
```
(ns coollang.repl
  (:require [coollang.parser :as parser]))
```

```
(defn repl []
  (do
    (print "MyCoolLang> ")
    (flush))
  (let [input (read-line)]
    (println (parser/parse input))
    (recur)))

(defn -main [& args]
  (println "Hello MyCoolLang!")
  (println "==================")
  (flush)
  (repl))
```



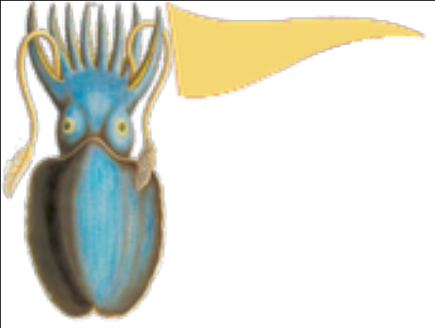
How to build a language?

```
(ns coollang.repl
  (:require [coollang.parser :as parser]))

(defn repl []
  (do
    (print "MyCoolLang> ")
    (flush))
  (let [input (read-line)]
    (println (parser/parse input))
    (recur)))

(defn -main [& args]
  (println "Hello MyCoolLang!")
  (println "==================")
  (flush)
  (repl))
```



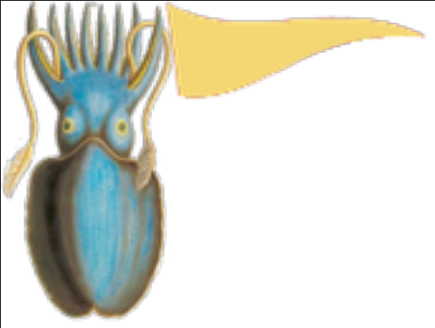
How to build a language?

```
(ns coollang.repl
  (:require [coollang.parser :as parser]))

(defn repl []
  (do
    (print "MyCoolLang> ")
    (flush))
  (let [input (read-line)]
    (println (parser/parse input))
    (recur)))

(defn -main [& args]
  (println "Hello MyCoolLang!")
  (println "==================")
  (flush)
  (repl))
```

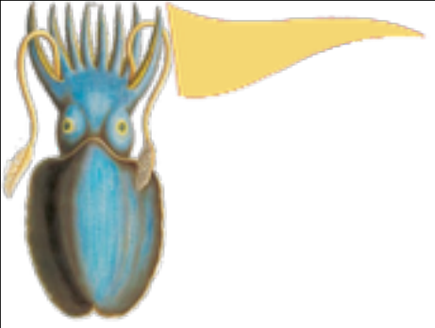


How to build a language?

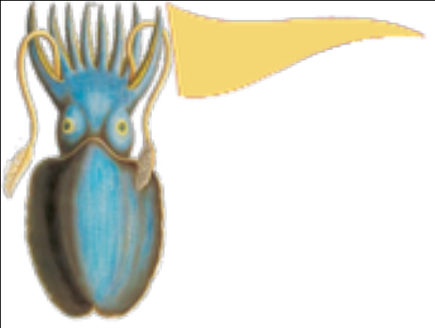
```
Hello MyCoolLang!
=================
MyCoolLang> + 1 2 3 4
10
MyCoolLang> 1 2 3
[1 2 3]
MyCoolLang> 1
1
```



Days Passed

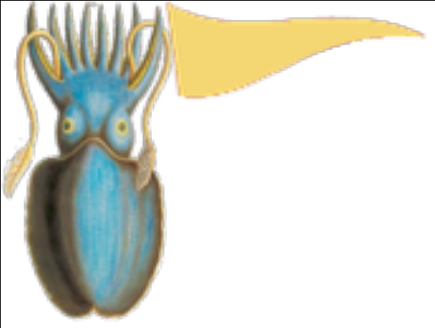


Days Passed

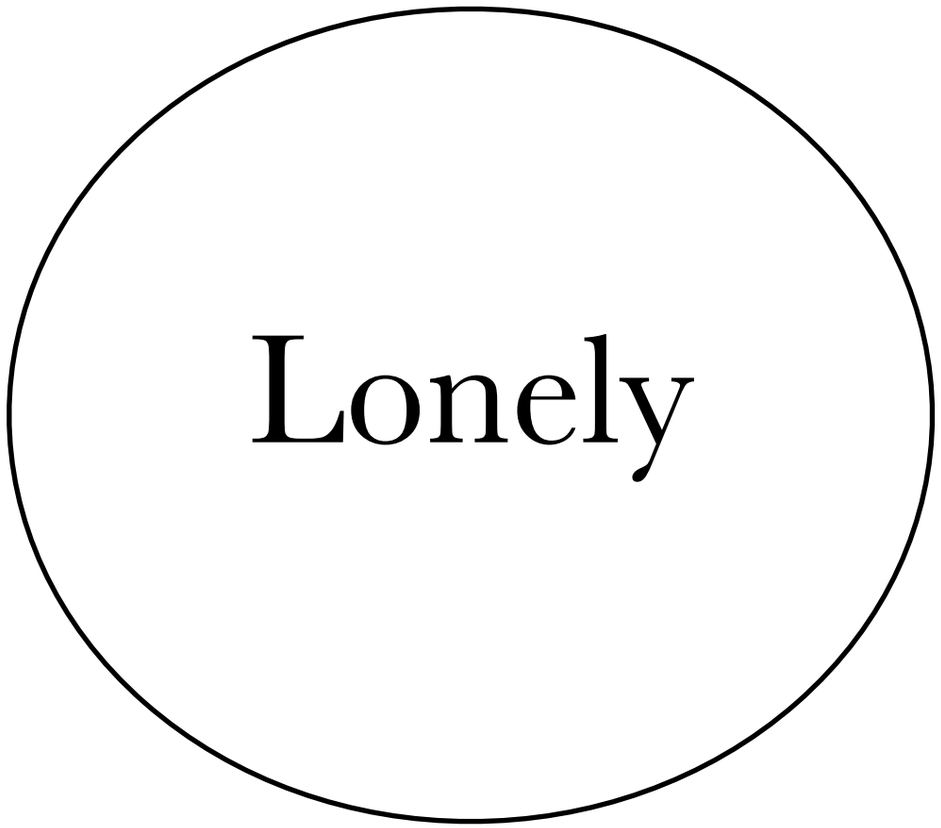


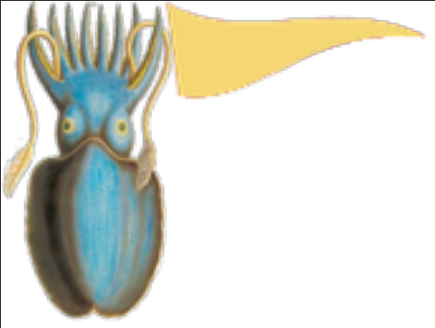
Days Passed

```
def defn
if or and not
import
println
get
do
sleep
first
swap! reset!
fn
```



Days Passed

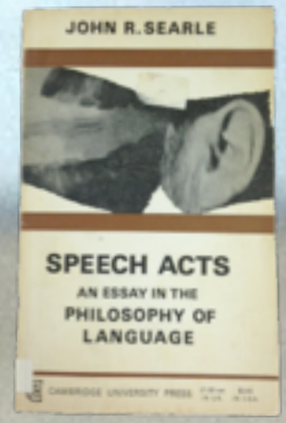




Days Passed

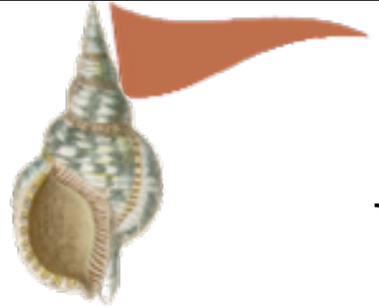
One Day

One Day

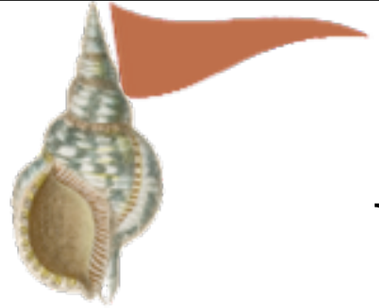


Speech Acts
John R. Searle

Elephant 2000
John McCarthy



What does it mean to speak?



What does it mean to speak?

"Pass the salt."

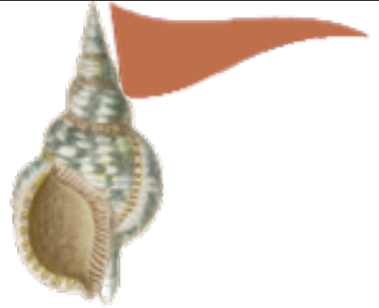
What does it mean to speak?

"Pass the salt."

Utterance: Sounds

Meaning: Higher order than language (English/French)

When a person hears the sentence, it is interpreted as a command.

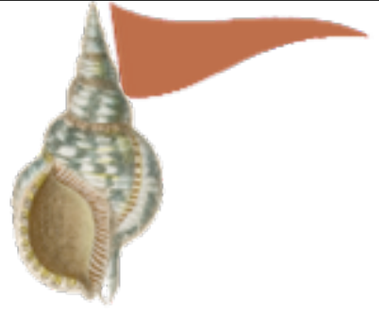


Speech Acts

Illocutionary Acts

J.L. Austin "How to Do Things with Words" (1962)
John R. Searle "Speech Acts" (1969)

- Assert - "It is hot."
- Command/ Request - "Pass the seaweed"
- Query - "Do you see a coconut?"



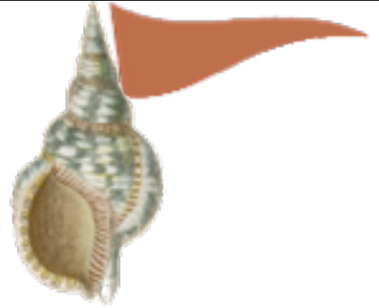
Speech Acts

Perlocutionary Acts

John R. Searle "Speech Acts" (1969)

Concerned with the **effects** the acts have on the actions, thoughts and beliefs, etc of the hearers.

- Persuade
- Convince
- Alarm
- Scare

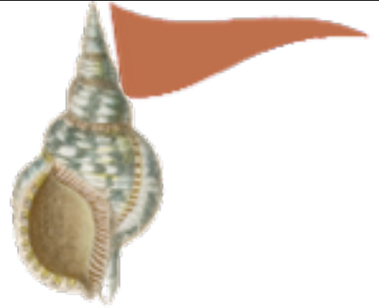


Elephant 2000

John McCarthy(1992)

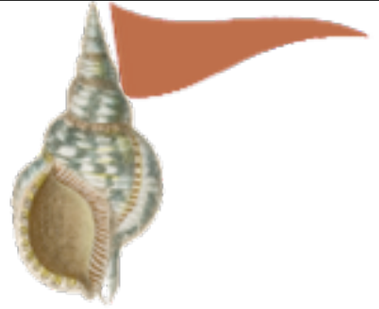
Ideas about Programming Languages

Communication inputs and outputs are in an I-O language whose sentences are meaningful speech acts.



Elephant 2000

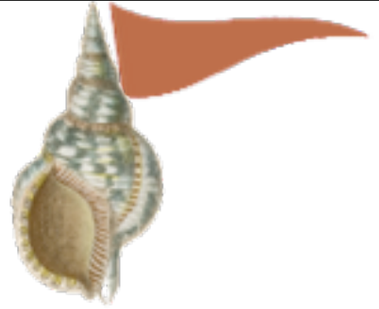
What can computer language use of speech acts learn from the extensive work of philosophers?



Elephant 2000

What can computer language use of speech acts learn from the extensive work of philosophers?

What light can be shed on philosophical problems when we consider how we want to use speech acts with computers?



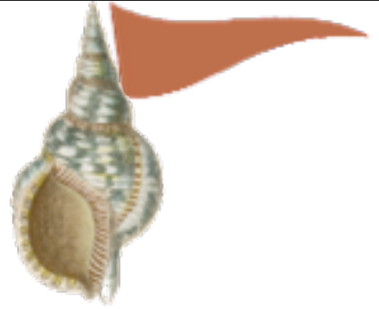
Elephant 2000

What can computer language use of speech acts learn from the extensive work of philosophers?

What light can be shed on philosophical problems when we consider how we want to use speech acts with computers?

Speech acts are necessary in the *common sense informatic situation* in which people interact to achieve goals.

- Independent of being human
- Martians or Robots would also require speech acts.



Why not talk to the REPL with Speech Acts?

Asserting

Illocutionary Speech Act

```
> assert sunny true
```

Answering Queries

Illocutionary Speech Act

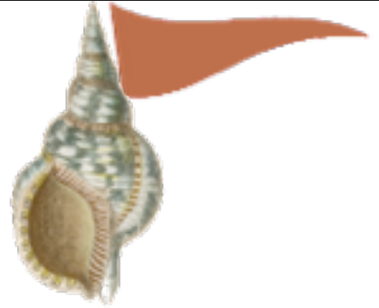
```
> query value sunny
true
```

Accepting Requests

Illocutionary Speech Act

"Pass the salt."

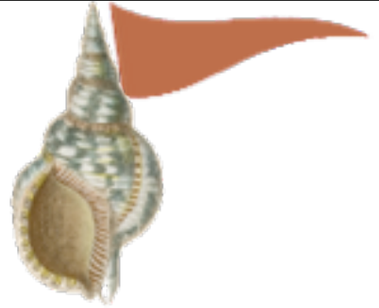
Accepting a request creates an internal commitment that is performed at a future time



Accepting Requests

Datatype commitment: *pass-salt

internal list of commitments

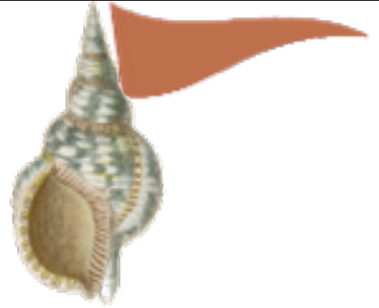


Accepting Requests

Datatype commitment: *pass-salt

internal list of commitments

Cron like process that continually checks to see if there are any commitments that need to be performed



Accepting Requests

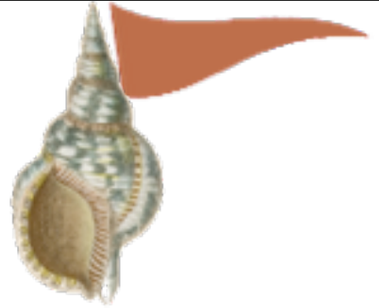
Datatype commitment: *pass-salt

internal list of commitments

Cron like process that continually checks to see if there are any commitments that need to be performed

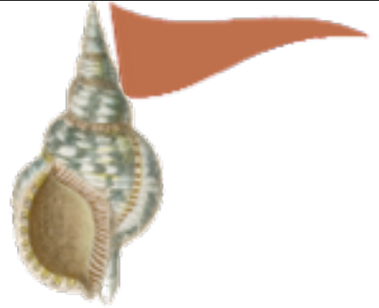
request: request commitment function

```
request *pass-salt fn [] :salt
;=> speech_acts.Commitment
```



Beliefs

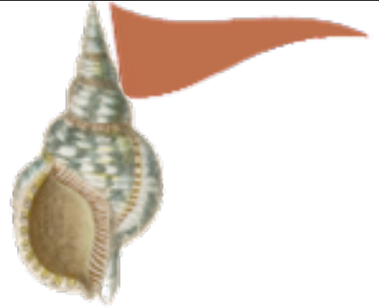
```
I have beliefs too>
```



Beliefs

```
I have beliefs too>
```

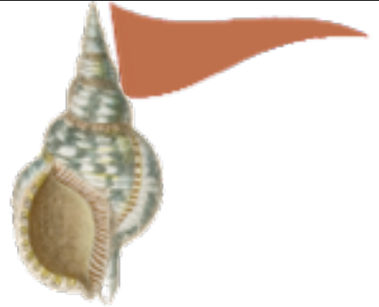
John McCarthy - Ascribing Mental Qualities to Machines



Beliefs

Why?

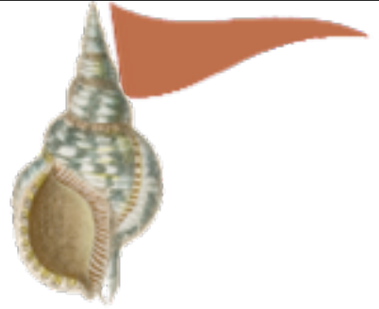
"Although we may know the program, its state at a given moment is usually not directly observable, and the facts we can obtain about its current state may be more ***readily expressed*** *by ascribing certain beliefs and goals than in any other way."*



Beliefs

Why?

"The belief and goal structure is likely to be close to the structure the designer of the program had in mind, and it may be ***easier to debug*** *..."*

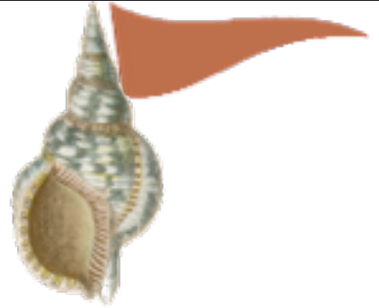


Beliefs

Datatype belief: `#nice-day`

Belief is described by a predicate function

When the predicate function evals to true, the belief is “held”



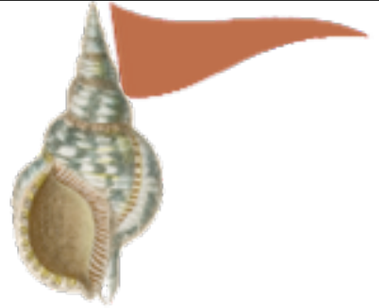
Convince

Perlocutionary Speech Act: Convince

Creates an internal belief that has a human readable description and a predicate function, which when it evals to true, is said to be “held”

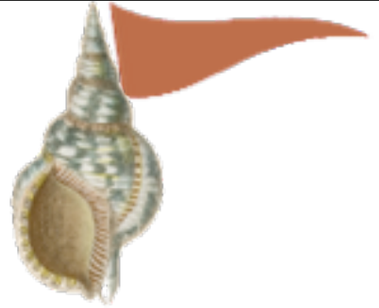
```
convince - convince belief string predicate-function
```

```
> convince #niceday "It is a nice day." fn [] = sunny true

[#speech_acts.Belief{:str It is a nice day., :fn #<user
$eval1465$fn__1466 user$eval1465$fn__1466@3de8754c>}]
```



Requests with Beliefs

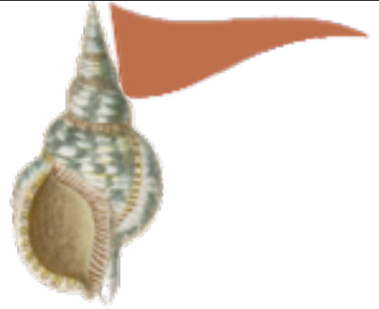
```
>assert sunny false
#'user/sunny
```



Requests with Beliefs

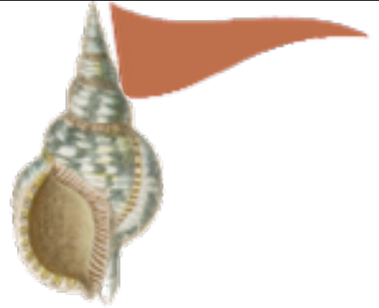
```
>assert sunny false
#'user/sunny
```

```
>convince #nice-day "It is a nice day." fn [] = sunny true
#speech_acts.Belief{:str It is a nice day., :fn #<user$eval1473$fn__1474
user$eval1473$fn__1474@2af5e863>}
```



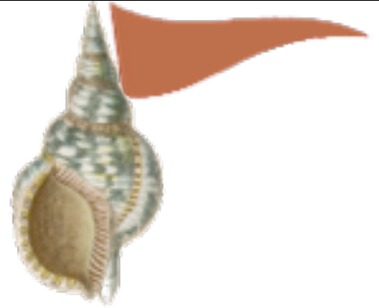
Requests with Beliefs

```
>assert sunny false
#'user/sunny

>convince #nice-day "It is a nice day." fn [] = sunny true
#speech_acts.Belief{:str It is a nice day., :fn #<user$eval1473$fn__1474
user$eval1473$fn__1474@2af5e863>}

>request *open-window when #nice-day fn [] println "Opened the window"
#speech_acts.Commitment
```



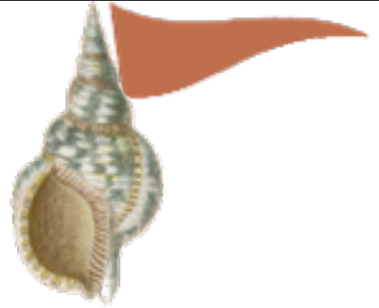
Requests with Beliefs

```
>assert sunny false
#'user/sunny

>convince #nice-day "It is a nice day." fn [] = sunny true
#speech_acts.Belief{:str It is a nice day., :fn #<user$eval1473$fn__1474
user$eval1473$fn__1474@2af5e863>}

>request *open-window when #nice-day fn [] println "Opened the window"
#speech_acts.Commitment

>query request-is-done *open-window?
false
```



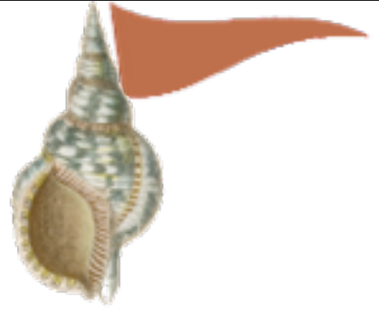
Requests with Beliefs

```
>assert sunny false
#'user/sunny

>convince #nice-day "It is a nice day." fn [] = sunny true
#speech_acts.Belief{:str It is a nice day., :fn #<user$eval1473$fn__1474
user$eval1473$fn__1474@2af5e863>}

>request *open-window when #nice-day fn [] println "Opened the window"
#speech_acts.Commitment

>query request-is-done *open-window?
false

>assert sunny true
#'user/sunny
Opened the window
```



Requests with Beliefs

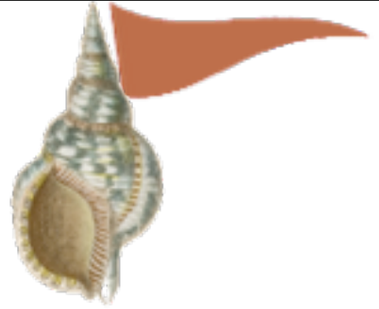
```
>assert sunny false
#'user/sunny

>convince #nice-day "It is a nice day." fn [] = sunny true
#speech_acts.Belief{:str It is a nice day., :fn #<user$eval1473$fn__1474
user$eval1473$fn__1474@2af5e863>}

>request *open-window when #nice-day fn [] println "Opened the window"
#speech_acts.Commitment
>query request-is-done *open-window?
false

>assert sunny true
#'user/sunny
Opened the window

>query request-is-done *open-window?
true
```



Talk to me

Tell me what you believe

Hello Babar

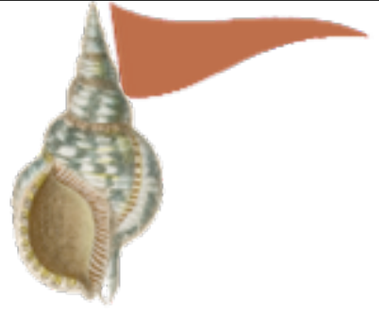


Talk to me

Tell me what you believe

```
Hello Babar
```

Demo

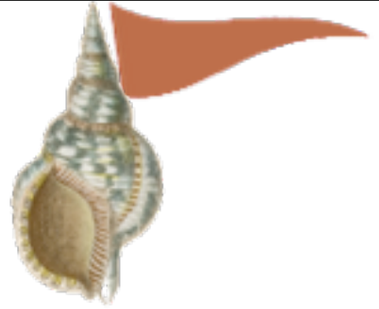


Talk to me

Ask me Questions

Why can't the REPL ask me questions?

```
fn [] + 10 x
```



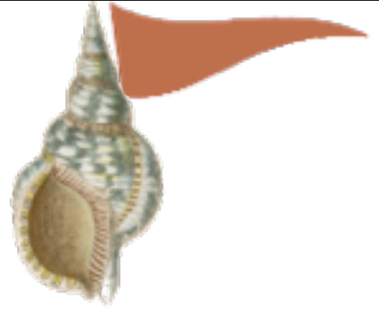
Talk to me

Ask me Questions

Why can't the REPL ask me questions?

```
fn [] + 10 x
```

```
ask-config true
```



Talk to me

Ask me Questions

Why can't the REPL ask me questions?

```
fn [] + 10 x
```

```
ask-config true
```

Demo

So happy to have someone to talk with!!!

A Ship?

NO!

Don’t miss the boat

Speech Acts with Drones

Boat recognition

Use OpenCV to recognize boat

Watch for the boat

spot_boat.babar

```
speak-config true.

import "clj-drone.core".
import "clj-drone.navdata".
```

```
(drone-initialize).
(drone-init-navdata).

assert navdata (get-nav-data :default).
assert drone-navdata [key] get @navdata key.
assert navdata-equal [key val] = (drone-navdata key) val.
assert navdata-gt [key val] > (drone-navdata key) val.
```

spot_boat.babar

```
speak-config true.

import "clj-drone.core".
import "clj-drone.navdata".

(drone-initialize).
(drone-init-navdata).

assert navdata (get-nav-data :default).
assert drone-navdata [key] get @navdata key.
assert navdata-equal [key val] = (drone-navdata key) val.
assert navdata-gt [key val] > (drone-navdata key) val.
```

spot_boat.babar

```
speak-config true.

import "clj-drone.core".
import "clj-drone.navdata".

(drone-initialize).
(drone-init-navdata).

assert navdata (get-nav-data :default).
assert drone-navdata [key] get @navdata key.
assert navdata-equal [key val] = (drone-navdata key) val.
assert navdata-gt [key val] > (drone-navdata key) val.
```

spot_boat.babar

```
assert see-boat false.

convince #spotboat "I see a boat!" fn [] = see-boat true.
convince #flying "I am flying" fn [] or (navdata-equal :control-state :flying)
                                        (navdata-equal :control-state :hovering).
convince #high-enough "I am signaling the boat" fn [] (navdata-gt :altitude 1.3).


request *take-off when #spotboat fn [] (drone :take-off).
request *cruising-alt when #flying until #high-enough fn [] (drone :up 0.3).
request *land when #high-enough fn [] do (drone :hover)
                                         (drone :anim-double-phi-theta-mixed)
                                         (sleep 5000)
                                         (drone :land).
```

spot_boat.babar

```
assert see-boat false.

convince #spotboat "I see a boat!" fn [] = see-boat true.
convince #flying "I am flying" fn [] or (navdata-equal :control-state :flying)
                                        (navdata-equal :control-state :hovering).
convince #high-enough "I am signaling the boat" fn [] (navdata-gt :altitude 1.3).

request *take-off when #spotboat fn [] (drone :take-off).
request *cruising-alt when #flying until #high-enough fn [] (drone :up 0.3).
request *land when #high-enough fn [] do (drone :hover)
                                         (drone :anim-double-phi-theta-mixed)
                                         (sleep 5000)
                                         (drone :land).
```

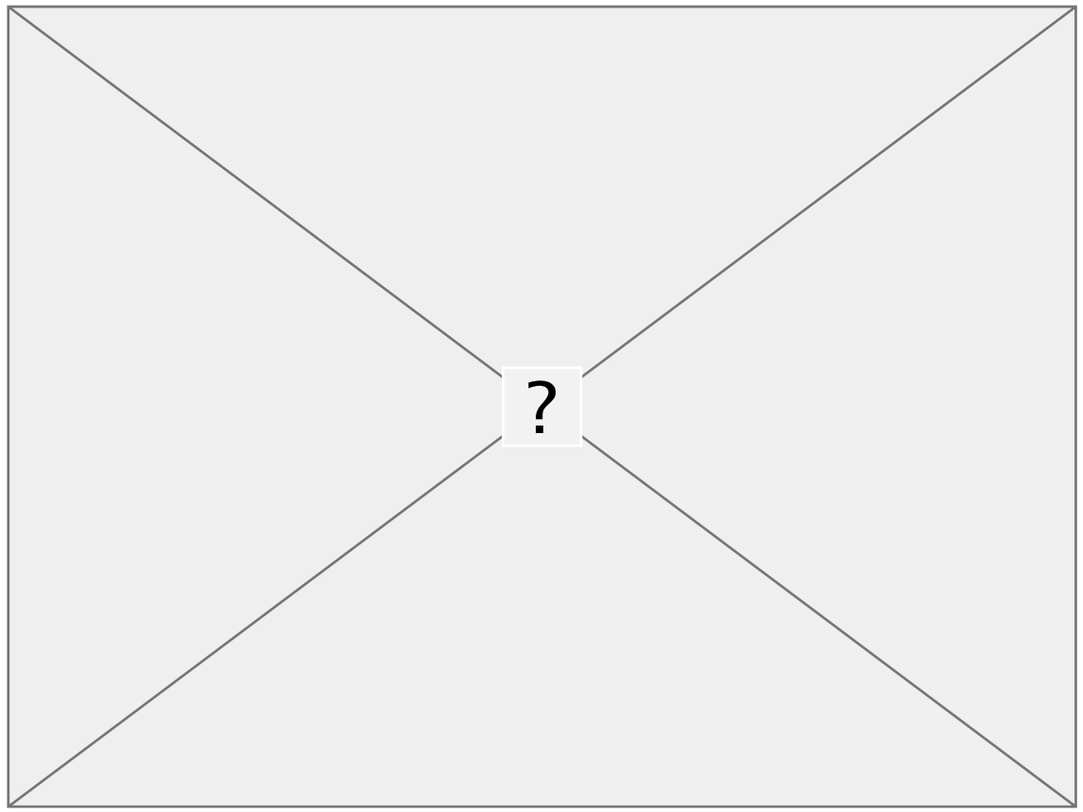
spot_boat.babar

```
assert see-boat false.

convince #spotboat "I see a boat!" fn [] = see-boat true.
convince #flying "I am flying" fn [] or (navdata-equal :control-state :flying)
                                        (navdata-equal :control-state :hovering).
convince #high-enough "I am signaling the boat" fn [] (navdata-gt :altitude 1.3).
```

```
request *take-off when #spotboat fn [] (drone :take-off).
request *cruising-alt when #flying until #high-enough fn [] (drone :up 0.3).
request *land when #high-enough fn [] do (drone :hover)
                                         (drone :anim-double-phi-theta-mixed)
                                         (sleep 5000)
                                         (drone :land).
```

Demo!

```
read "examples/drone/spotboat.babar"
```

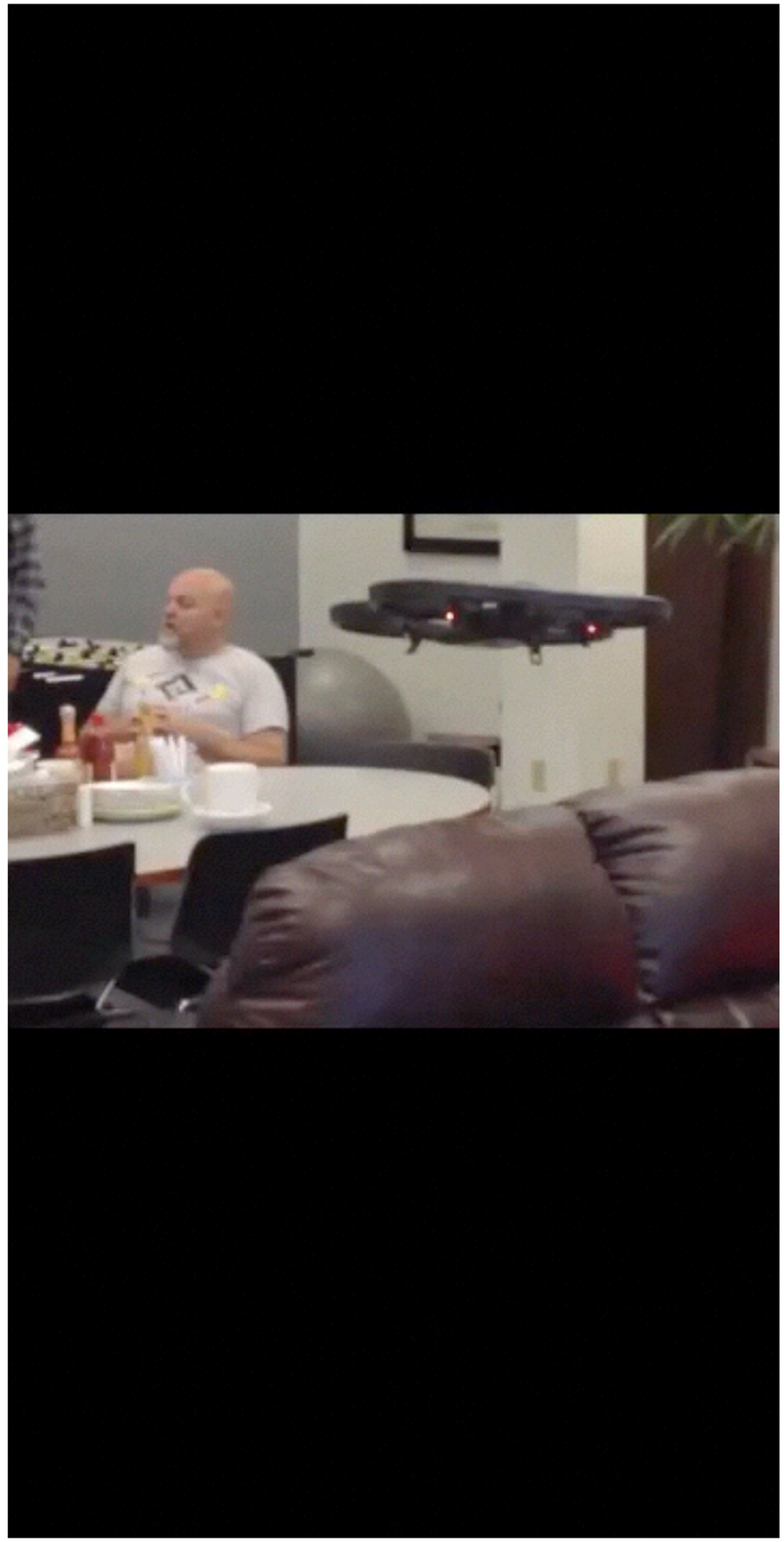


with Bling

Demo!

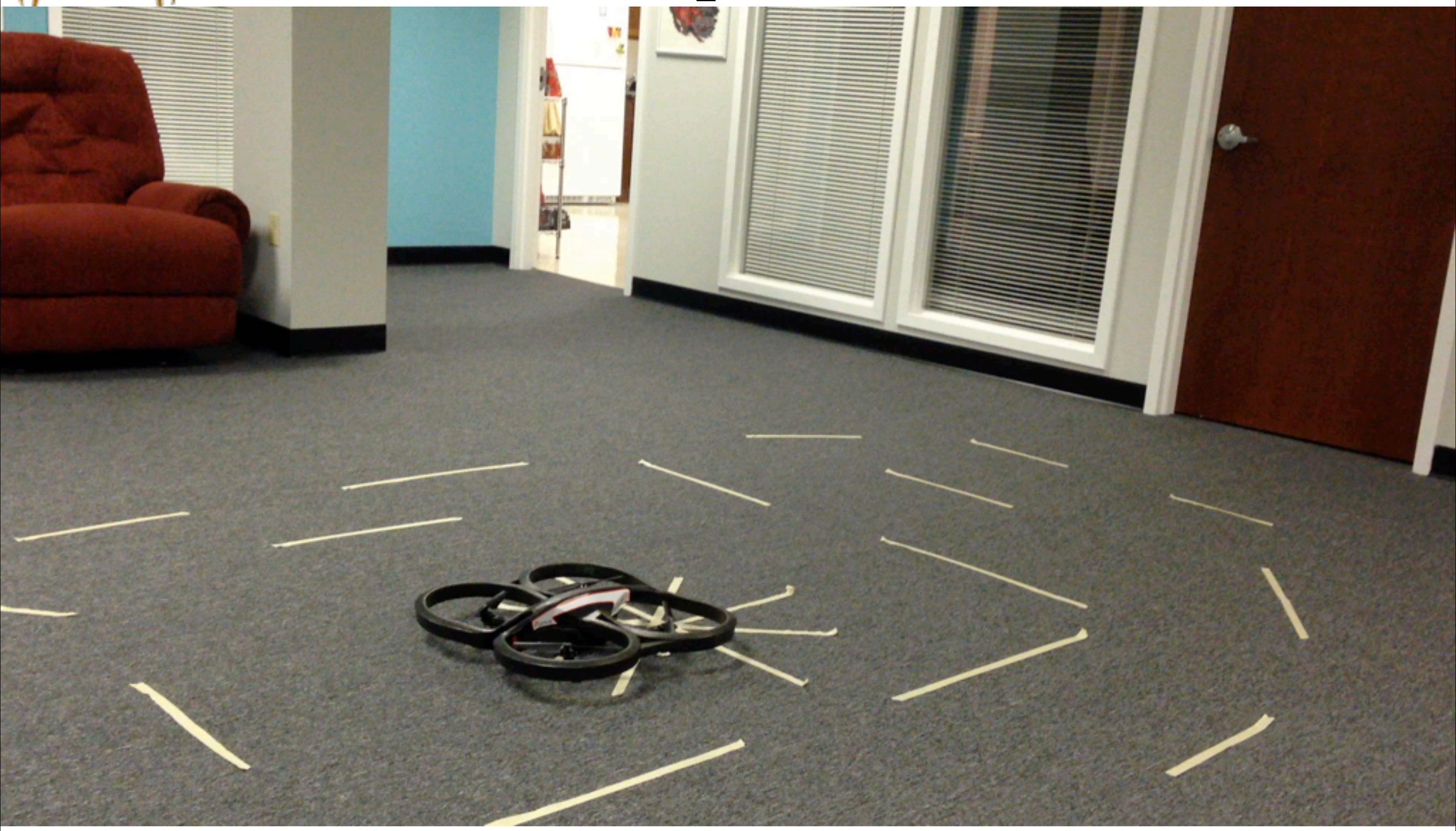
Safety First

How Not to Catch a Drone



How to
to Catch a Drone

Backup Video



Rescued!!

Thank you.

That’s a speech act too

Closing/ Recap

Instaparse is a beautiful parser

https://github.com/Engelberg/instaparse

Closing/ Recap

Computer Science can learn from Philosophy

Philosophy can learn from Computer Science

Speech Acts are about the meaning behind our communication - whether that is with another human or a machine

Closing/ Recap

PPPJ 2013 full research paper

Feel Different on the Java Platform:
The Star Programming Language

Frank McCabe

Michael Sperber

http://sourceforge.net/projects/starlanguage/

- Full-scale programming language that incorporates a simplified model of speech act theory as a basis for programming actors
- Star has actors and speech actions

Closing/ Recap

You should read John McCarthy's papers

http://www-formal.stanford.edu/jmc/elephant/elephant.html

http://www-formal.stanford.edu/jmc/ascribing/ascribing.html

Playing with Drones and Clojure is great fun!

https://github.com/gigasquid/babar

https://github.com/gigasquid/clj-drone

References

http://farm2.staticflickr.com/1277/870683349_d0c4de7679.jpg - clock
http://farm7.staticflickr.com/6129/5993988615_1fd3d2c530_b.jpg - boat
http://www.ll-flightcases.com/353/storm-case-im2450
http://farm3.staticflickr.com/2423/4052389324_0d15da4b8f_z.jpg - computer
http://farm4.staticflickr.com/3092/2351384746_e981ed5fca_z.jpg?zz=1 - margartia
http://farm9.staticflickr.com/8125/8653649155_b3b48a05b3_z.jpg storm clouds
A Cabinet of Natural Curiosities (all the cool little critters & seamonster)
http://www.strangescience.net/stsea2.htm#searhino
http://farm4.staticflickr.com/3159/2580224978_faa5bd2724_b.jpg - beach
http://www.amazon.com/Nature-Power-55020-18-Watt-Monocrystalline/dp/B007G83WFO/ref=pd_sim_sbs_misc_8
http://farm1.staticflickr.com/6/75708748_24b1b2499d_z.jpg?zz=1 - tree house
http://farm2.staticflickr.com/1184/5110038952_c05dcd1a0d_z.jpg - suitcase
http://farm4.staticflickr.com/3096/2760110362_a9a719884c_z.jpg - salt
http://farm9.staticflickr.com/8506/8400150382_23b96103cf_z.jpg - question
http://farm3.staticflickr.com/2057/2233280919_064f535dee.jpg - sunny beach
http://farm2.staticflickr.com/1100/683560046_b43e4a4333_z.jpg - shrimp boat
http://farm1.staticflickr.com/2/2071546_a582ea634d_o.jpg - sea

.