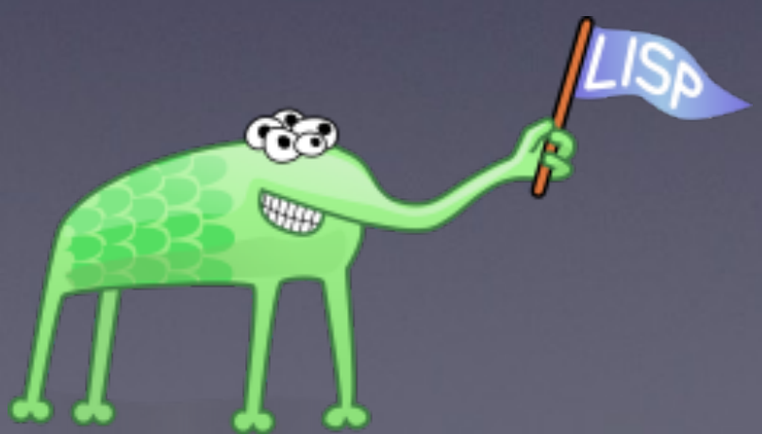
REPL 指向

第 13 回 #渋谷java

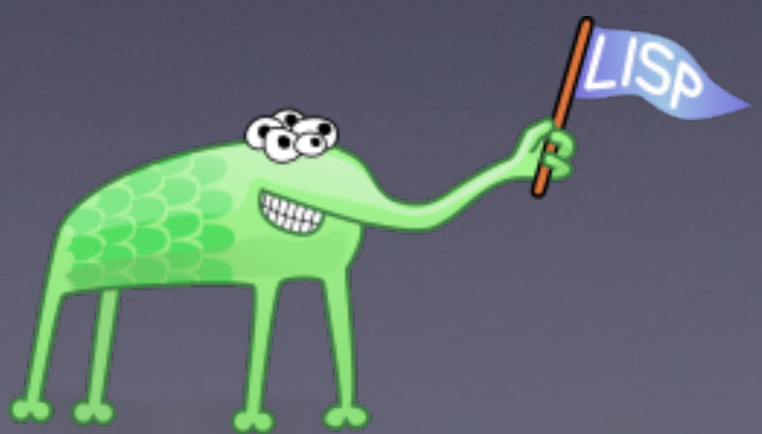
@_ayat_p / Cybozu Startups, Inc.



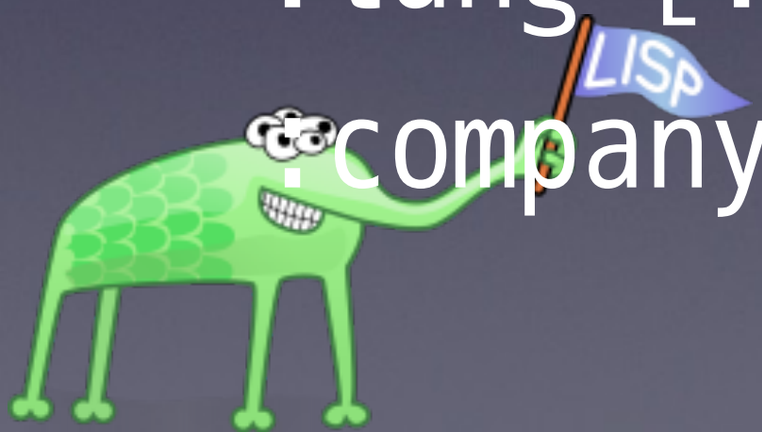
REPL 指向至高

第 13 回 #渋谷java

@_ayat_p / Cybozu Startups, Inc.



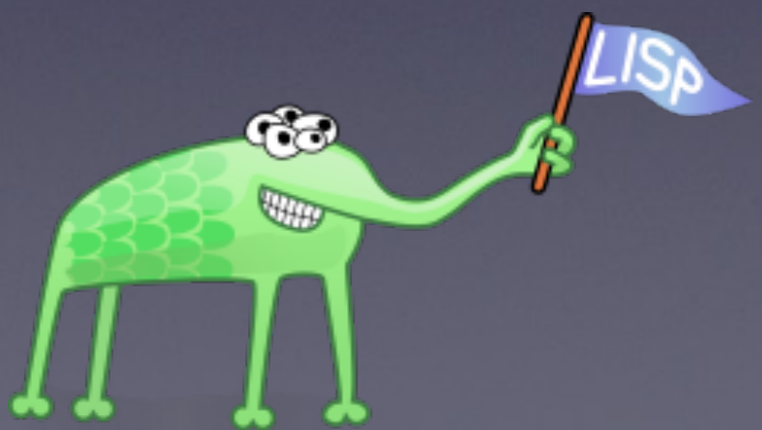
```
(def _ayato_p
  {:name "あやぴー"
   :lang [:clojure]
   :company "Cybozu Startups, Inc."})
```



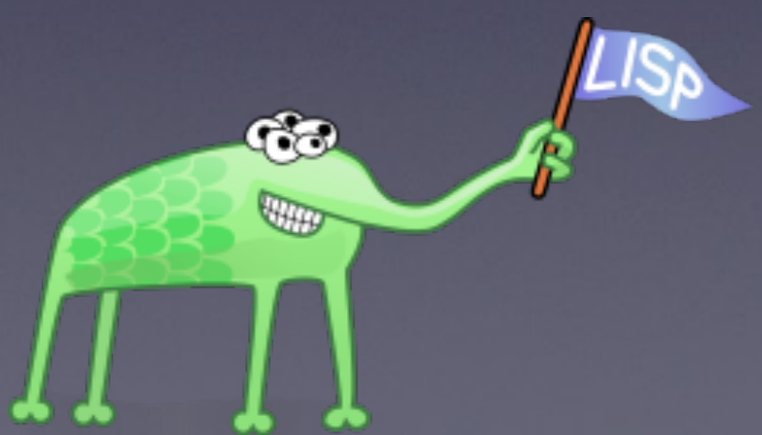
最近 Clojure しか書けなくなりました

```
(def ayatsuri
  {:name "あやつり"
   :lang [:clojure]
   :company "Cybozu Startups, Inc."})
```

はじめに



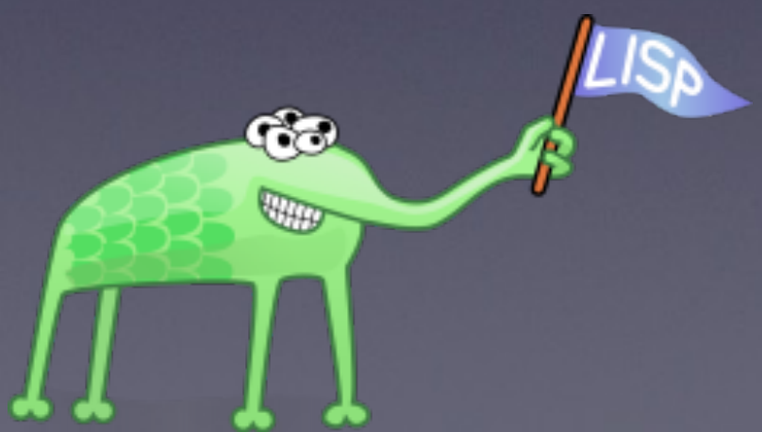
"REPL の話をします"



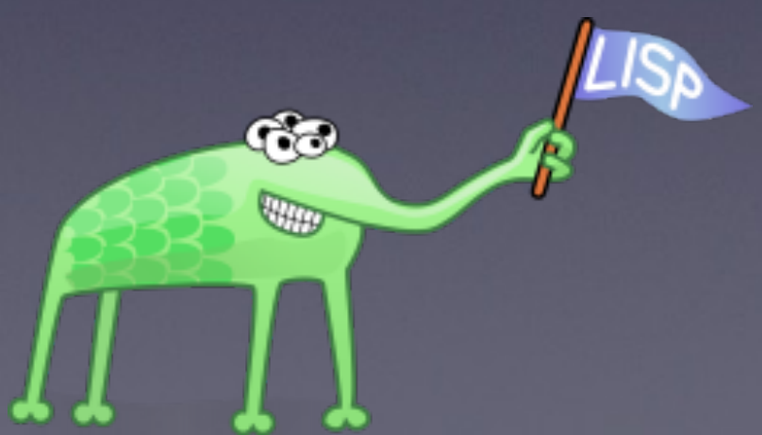
セッション枠(20分) **参加者**

セッション枠(20分) 参加者 **ayato_p**

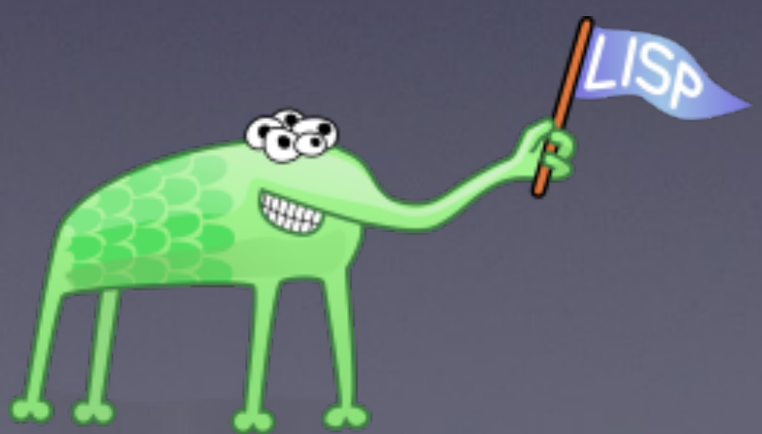
REPL の話します



とは言ったけど



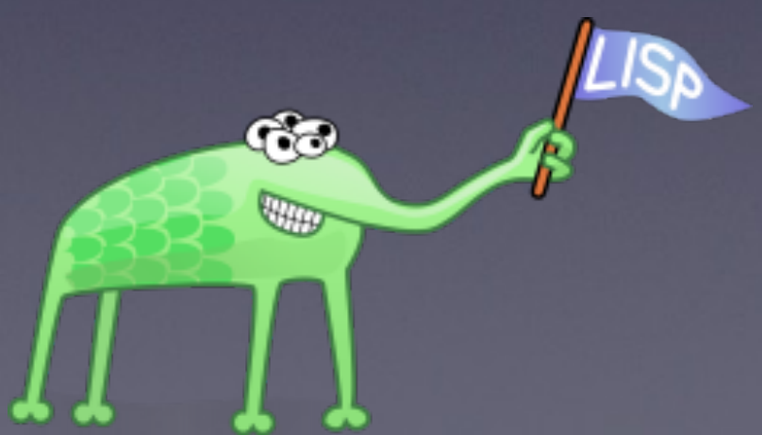
"Java の"とは一言も言ってない :P



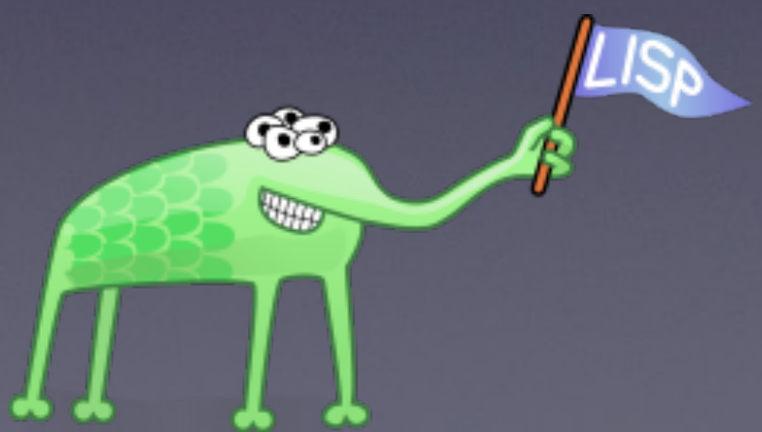
"Clojureの"REPLの話をします

"Java"のことは一言も言ってない :P

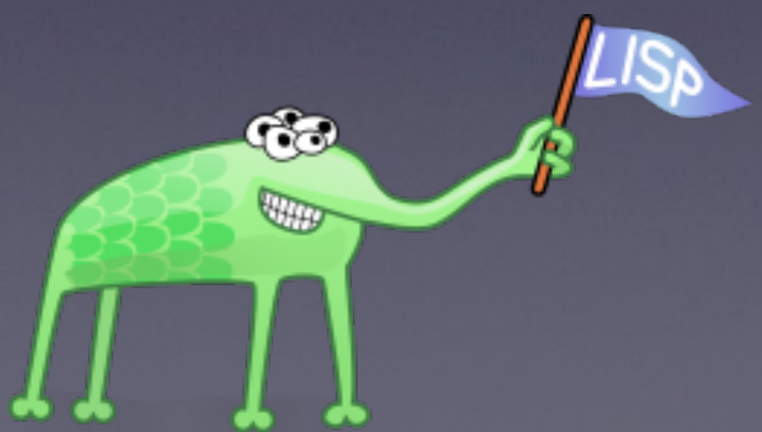
今日話すこと



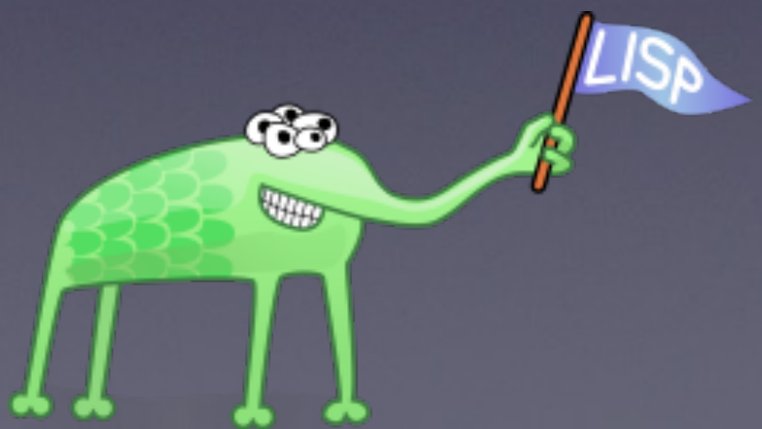
- 改めて REPL とは
- Clojure における REPL とは
- REPL 駆動開発



改めて REPL とは

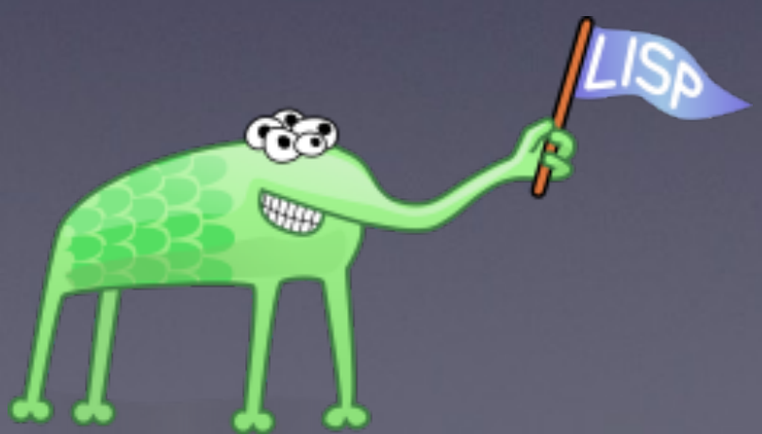


- 最近の言語はだいたい備えている
- Ruby -> irb, Python -> ipython
- Kotlin, Scala など JVM 言語でも!
  (そろそろ Java でも !?)



- Read-Eval-Print Loop

- 読んで、評価して、出力する繰り返し



#とは

- Read-Eval-Print Loop
- 読んで、評価して、出力する繰り返し

```
# e.g.) Ruby REPL

irb(main):001:0> def hello
irb(main):002:1>   "Hello, world"
irb(main):003:1> end
=> nil
irb(main):004:0> hello
=> "Hello, world"
```

```
# e.g.) Ruby REPL
```

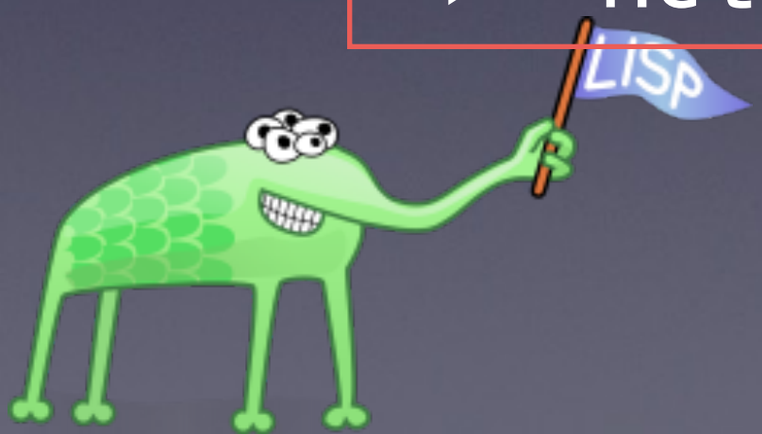
READ & EVAL

```
irb(main):001:0> def hello
irb(main):002:1>   "Hello, world"
irb(main):003:1> end
=> nil
```

READ & EVAL

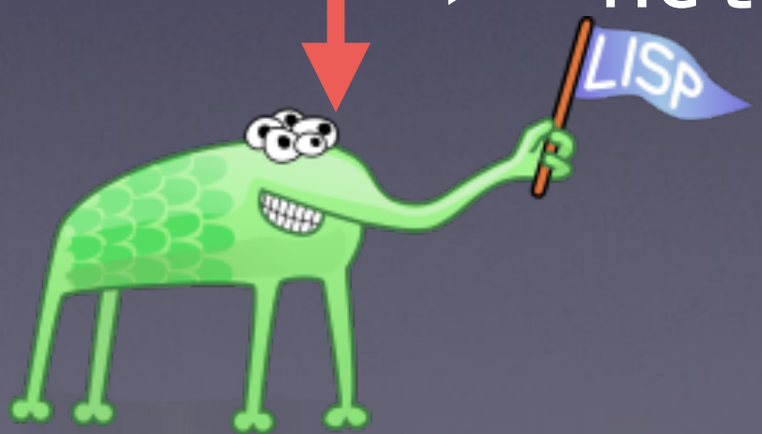
```
irb(main):004:0> hello
=> "Hello, world"
```

```
# e.g.) Ruby REPL

irb(main):001:0> def hello
irb(main):002:1>   "Hello, world"
irb(main):003:1> end                    PRINT
=> nil
irb(main):004:0> hello                  PRINT
=> "Hello, world"
```

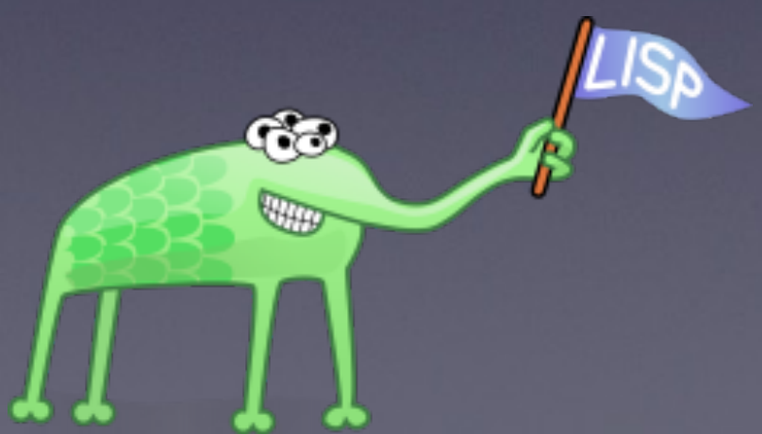


```
# e.g.) Ruby REPL

irb(main):001:0> def hello
irb(main):002:1>   "Hello, world"
irb(main):003:1> end
=> nil
irb(main):004:0> hello
=> "Hello, world"
```

LOOP



- 対話的に開発するためのツール
- 考えた機能を確認しながら実装できる
- （閉じた）REPL 環境に蓄積される
- たぶん便利(?)



- 対話的に開発するためのツール
- 考えた機能を確認しながら実装できる

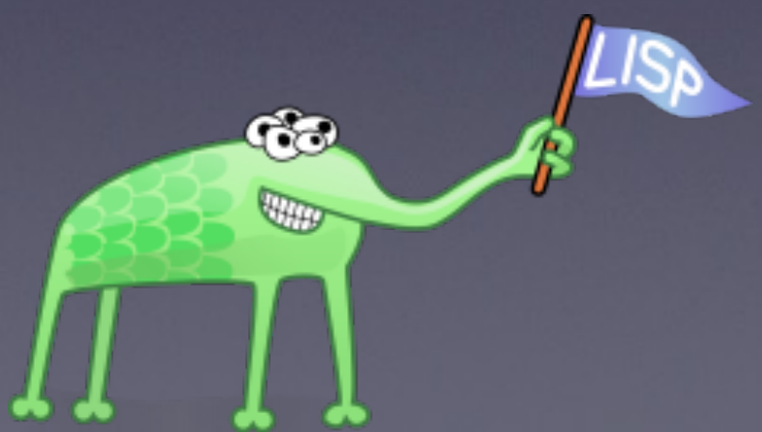
その REPL

- 慣れ親しんだ REPL 環境が蓄積されている

本当に便利ですか?

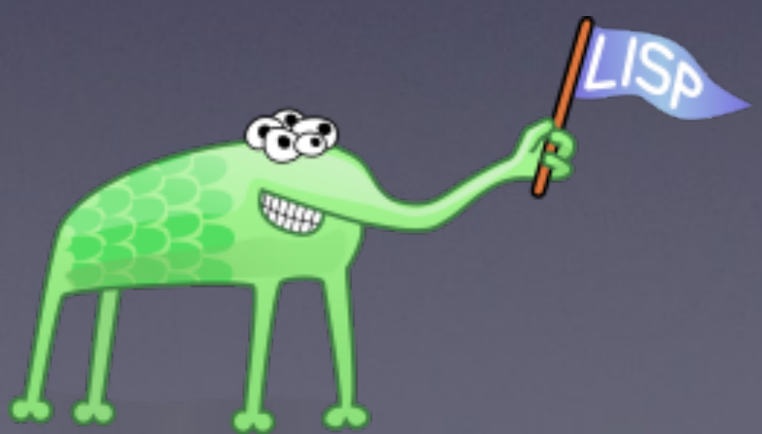
- たぶん便利(?)

Clojure における REPL とは



たぶんあなたはこう思っている?

「 "Clojure における REPL" ?
他の言語と同じだろ? 」



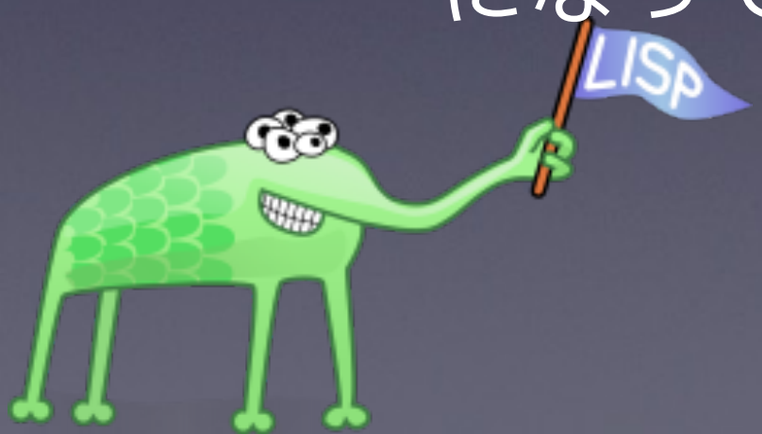
たぶんあなたはこう思っている?

違うんです

「 "Clojure における REPL" ?
他の言語と同じだろ? 」

Clojure の REPL

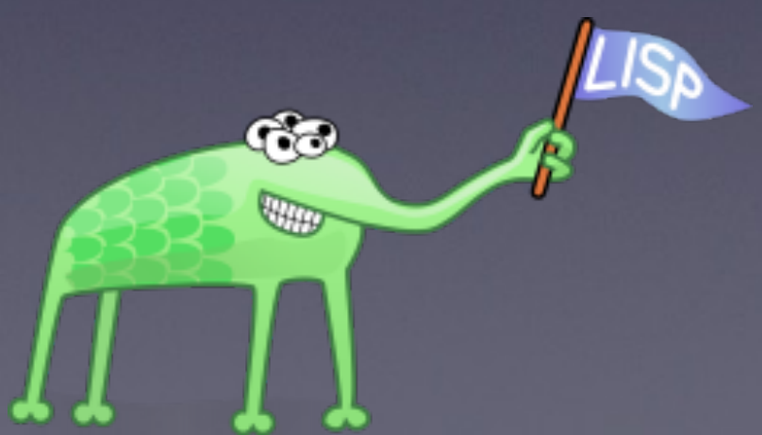
- Clojure は実行時に Clojure コンパイラの機能をフルで使える
- REPL 上での実行はファイルのロードと同じように動く
- テキストエディタ等と協調動作しやすいようになっている



違い

- ファイルベースのワークフロー
  (e.g. save -> auto-compile -> reload)
  が必要ない
- 実行中のプログラム環境を直接触れる
- ライブラリですら REPL 上で再定義可能
- 環境を触る機能が充実している
  (clojure.repl, tools.namespace とか)
- 本番環境ですら REPL を接続出来る

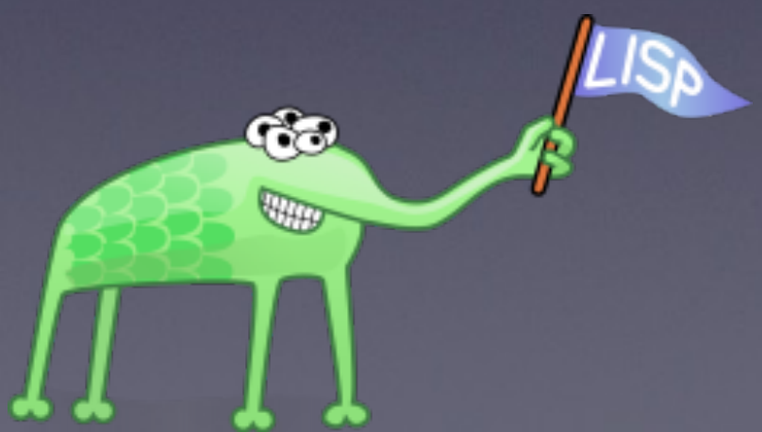
```
;; file
(ns example.core)

(defn inner-fn [])

(defn outer-fn []
  (inner-fn))
```




```
;; repl
e.c> (outer-fn)
=> nil

e.c> (defn inner-fn []
       "This is inner fn")
=> #'example.core/inner-fn

e.c> (outer-fn)
=> "This is inner fn"
```

こういう経験ありませんか?

- ファイル全体を更新したくないけど、一部変更を反映させたい
- 開発時に素早くマイグレーションしたい
- SQL をインクリメンタルに書きたい



- もっと早くトライアンドエラーしたい
- ブラウザの環境を使いたい (in AltJS)
- あの関数の名前が思い出せない
- ドキュメントだけだと分からないから試してみたい
- etc, etc...

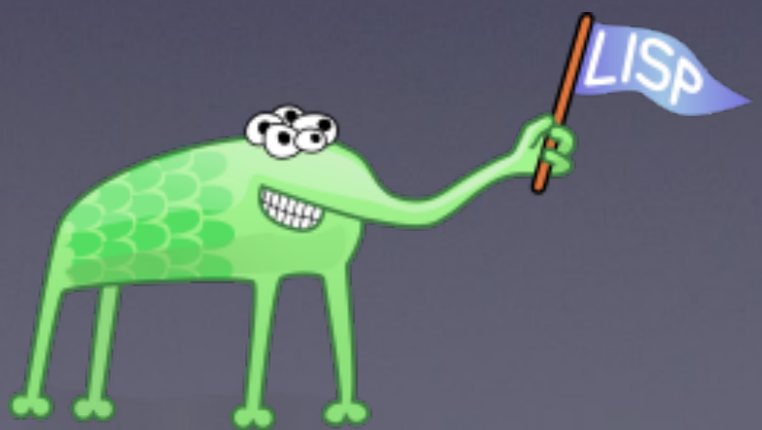
こういう経験ありませんか?

- ファイル全体を更新したくないけど、一部変更を反映させたい
- 素早くマイグレーションしたい
- SQLをコンスタントにメンタルに書きたい
- もっと早くトライアンドエラーしたい
- ブラウザの環境を使いたい (in AtJS)
- あのD関数の名前が思い出せない
- ドキュメントだけだと分からないから試してみたい
- etc, etc...

Clojureなら

簡単に解決可能

- REPL は Clojure での開発でなくてはならない存在
- むしろ REPL と適当なエディタで十分
  (Rich Hickey は実際に Emacs と事実上 REPL だけで開発しているとかなんとか)
- 逆に REPL ないとつらい

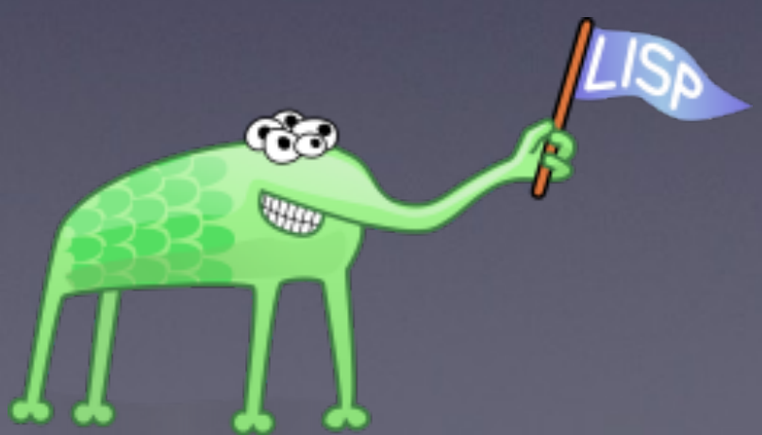


- REPL は Clojure での開発でなくては
  ならない存在

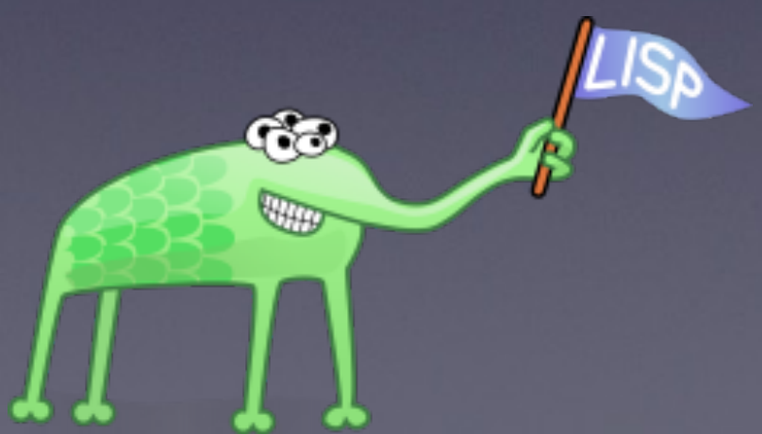
REPL が開発を加速させる

- むしろ REPL と適当なエディタで十分
  (Rich Hickey は実際にテキストエディタと REPL だけで開発しているとかなんとか)
- 逆に REPL ないとつらい

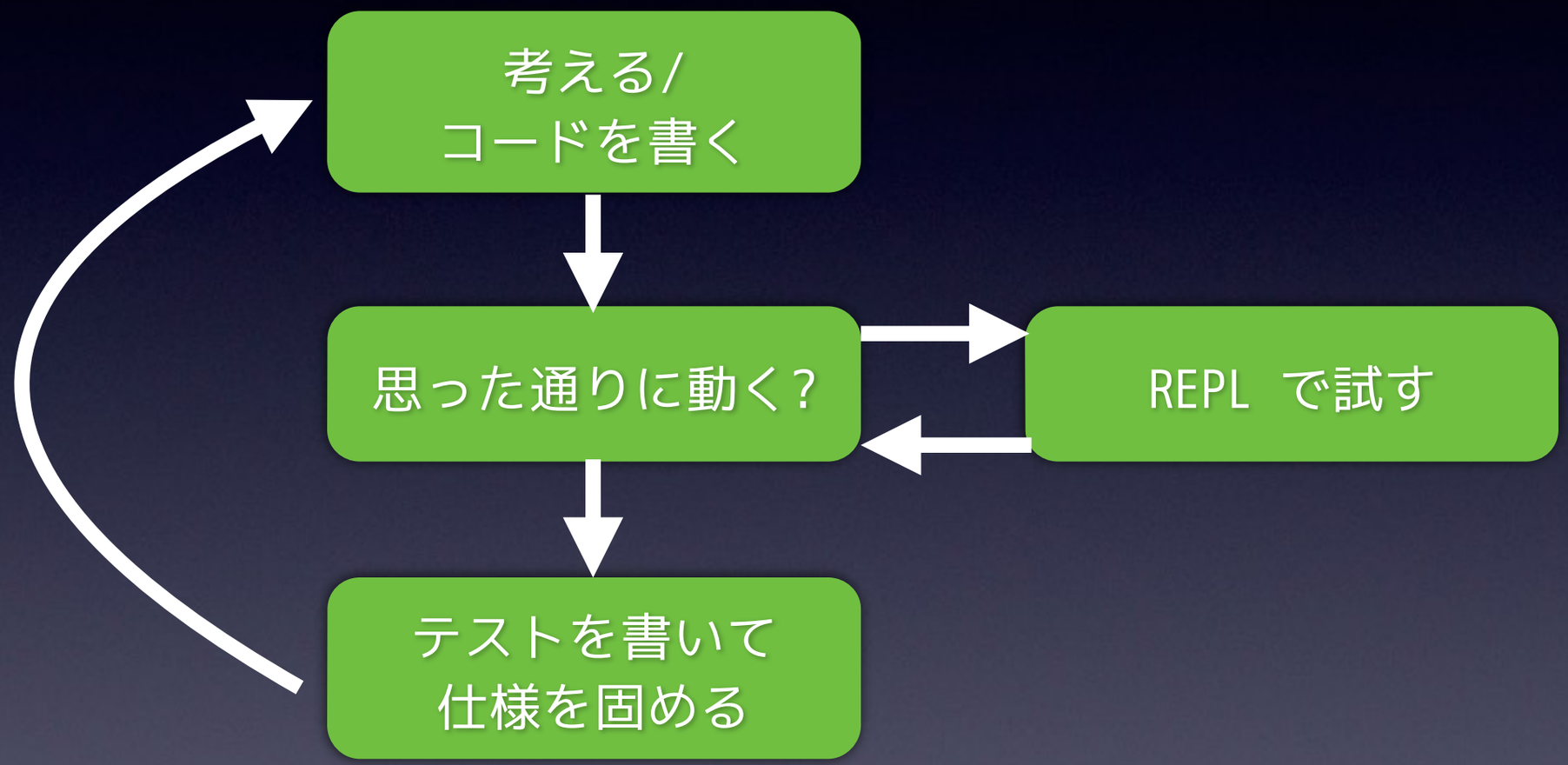
REPL 駆動開発

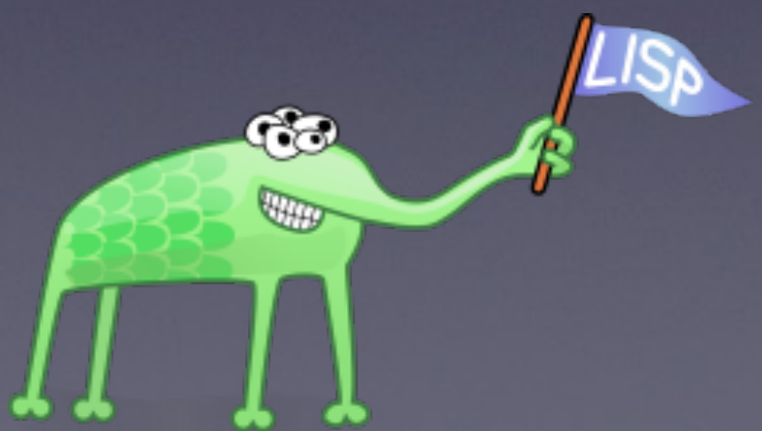


- REPL Driven Development
- REPL で素早いフィードバックを得る



ワークフロー



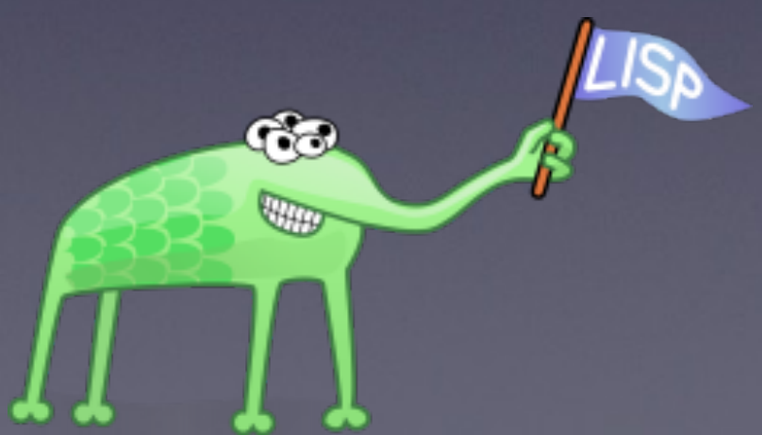


＊僕の日常です

- Clojure の特徴も相まって REPL 上であらゆる関数が簡単に実行可

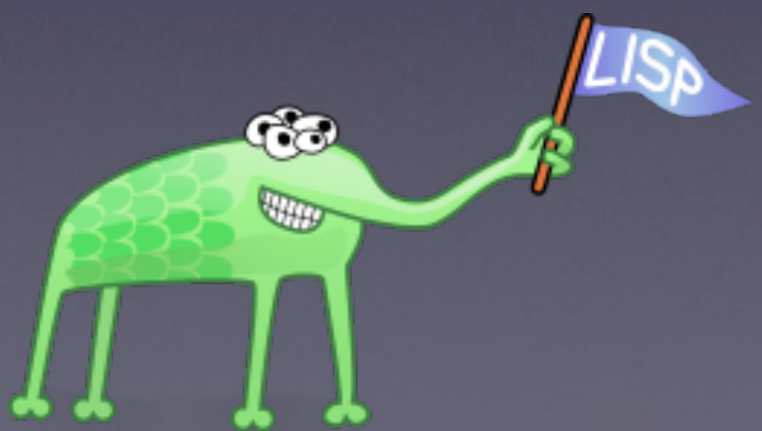
- REPL を使ってボトムアップ開発

- REPL で関数の定義やドキュメントが読める
  (悩んだら REPL に聞ける)



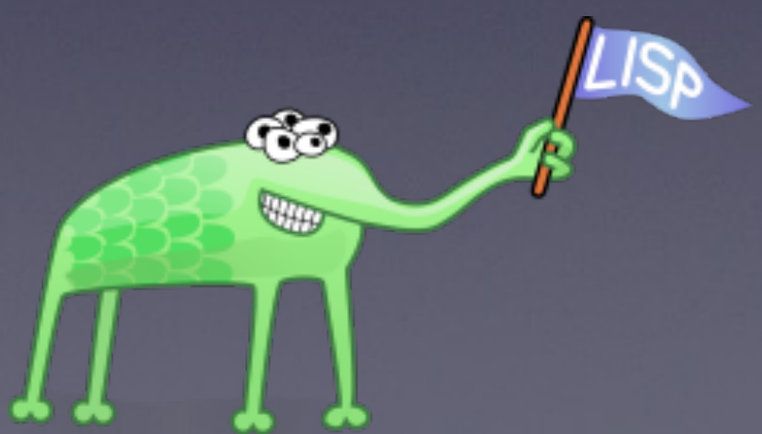
副産物的なもの

- REPL からあらゆるコマンドを実行可
  (サーバーを起動したり、マイグレーションしたり)
- JVM の起動オーバーヘッドがない

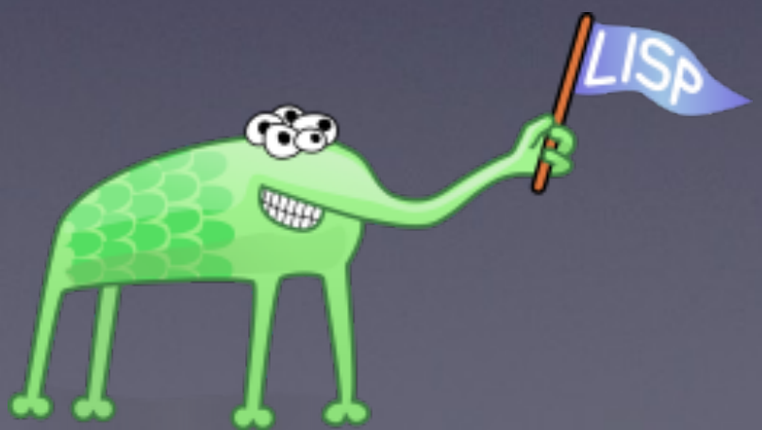


デメリット(?)

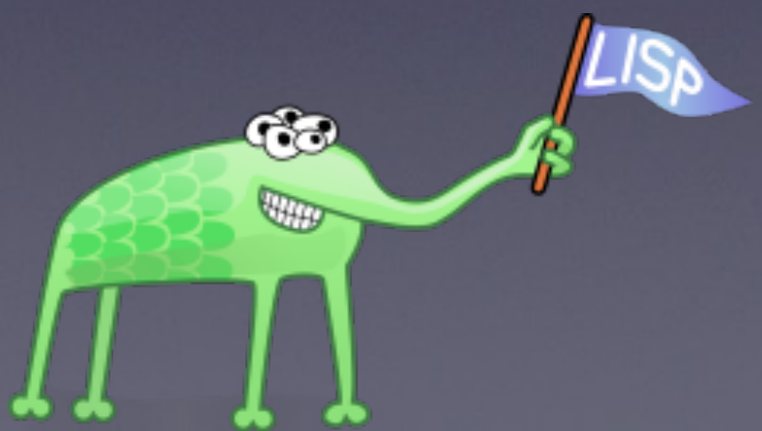
- まれに REPL 上の定義が残ってて誤動作するときがある
- テスト書きたくなくなる
  (だって動いてるし…)



まとめ



- REPL 駆動開発は楽しい
- Clojure の REPL は最高
- Clojure 最高



Enjoy Clojure