Adapting Clojure to an introductory CS classroom.

Elena Machkasova

University of Minnesota, Morris

Clojure/west, April 21 2015.

ClojurEd project

ClojurEd is a project at UMM: developing a beginner-friendly setup for Clojure.

ClojurEd project

ClojurEd is a project at UMM: developing a beginner-friendly setup for Clojure.

This is work in progress!
Lots still needs to be done before we can use Clojure in an introductory class.

Where are we coming from?

- Morris: rural town 3.5 hours from UMN-TC.
- UMM: small undergraduate-only liberal arts college.
- Undergraduate research.

Where are we coming from?

- Morris: rural town 3.5 hours from UMN-TC.
- UMM: small undergraduate-only liberal arts college.
- Undergraduate research.
- Use Lisp in an introductory CS class.

A programming language for introductory class

- **Use** a programming language in an introductory class.

A programming language for introductory class

- **Use** a programming language in an introductory class.
- **Teach** concepts and skills.

A programming language for introductory class

- **Use** a programming language in an introductory class.
- **Teach** concepts and skills.
- **Teach** CS-style problem solving.

A programming language for introductory class

- **Use** a programming language in an introductory class.
- **Teach** concepts and skills.
- **Teach** CS-style problem solving.
- The language needs to be simple, well-defined, and non-distracting (Felleisen et al, 2004).

A programming language for introductory class

- **Use** a programming language in an introductory class.
- **Teach** concepts and skills.
- **Teach** CS-style problem solving.
- The language needs to be simple, well-defined, and non-distracting (Felleisen et al, 2004).
- Dynamically typed.

A programming language for introductory class

- **Use** a programming language in an introductory class.
- **Teach** concepts and skills.
- **Teach** CS-style problem solving.
- The language needs to be simple, well-defined, and non-distracting (Felleisen et al, 2004).
- Dynamically typed.
- Functional languages offer focus on functions, modularity, abstraction.

A programming language for introductory class

- **Use** a programming language in an introductory class.
- **Teach** concepts and skills.
- **Teach** CS-style problem solving.
- The language needs to be simple, well-defined, and non-distracting (Felleisen et al, 2004).
- Dynamically typed.
- Functional languages offer focus on functions, modularity, abstraction.
- Understanding of higher order functions may be useful later for parallel programming and Javascript-based web dev.

A programming language for introductory class

- **Use** a programming language in an introductory class.
- **Teach** concepts and skills.
- **Teach** CS-style problem solving.
- The language needs to be simple, well-defined, and non-distracting (Felleisen et al, 2004).
- Dynamically typed.
- Functional languages offer focus on functions, modularity, abstraction.
- Understanding of higher order functions may be useful later for parallel programming and Javascript-based web dev.
- Interactive development promotes testing: REPL.

What Clojure can bring to the table in an intro class?

Currently use Racket (a Lisp), "How to design programs" curriculum.
Students are curious, would like to expand and apply what they learn.
Clojure offers:

- A large community that students can explore/benefit from: libraries, community-maintained docs, textbooks, forums, open source projects, meetups,....
- Concurrency.
- Integration with Java.
- Collections done right.

What's lacking

- Usability of error messages.
- Beginner-friendly IDE.
- Beginner-friendly project manager.
- Beginner-friendly graphical libraries.
- More uniform (from a beginner's standpoint) behavior of strings, collections vs sequences, etc.
- Beginner-friendly textbooks (focus on concepts, use language as a learning tool).

(Non)Usability of error messages

```
Exception in thread "main" java.lang.RuntimeException:
EOF while reading, starting at line 3,
compiling:(student/example.clj:4:1)
at clojure.lang.Compiler.load(Compiler.java:7137)
at clojure.lang.RT.loadResourceScript(RT.java:370)
at clojure.lang.RT.loadResourceScript(RT.java:361)....
```

- Phrased in terms of Java types and unfamiliar terminology.
- Java stack trace.
- Often misleading.
- Not well integrated with IDEs (LightTable does some line highlighting).

What do we do about error messages

- Our transformed message:

  ```
  Compilation error: end of file, starting at line 3,
  while compiling student/example.clj.
  Probably a non-closing parenthesis or bracket.
  ```

- Filter stack trace.
- Transform using regular expressions.
- Add preconditions to common functions (`map`, `filter`, etc).

Adding preconditions to common functions

Student code:

```
(map "inc" [1 2 3])
```

Original error message:

```
ClassCastException java.lang.String cannot be cast to
clojure.lang.IFn  clojure.core/map/fn--4245 (core.clj:257)
```

Our error message:

```
In function map, the first argument "inc" must be
a function but is a string.
```

Benefits and issues

- Add preconditions:

```
(defn map [argument1 & args]
 {:pre [(check-if-function? "map" argument1)
        (check-if-seqables? "map" args 2)]}
  (apply clojure.core/map argument1 args))
```

- Can determine specific arguments that have caused an error.
- Need to overwrite many functions (e.g. arithmetic functions).
- Need to load overwritten functions into student code and into REPL.
- Cannot overwrite all functions.
- Need to be careful.

Regular expression matching

Student's code (assuming no preconditions for `keep`):

```
(keep even? 6)
```

Original message:

```
java.lang.IllegalArgumentException: Don't know
how to create ISeq from: java.lang.Long
```

Our message:

```
Don't know how to create a sequence from a number.
```

Rephrase the message (if needed), substitute type names.

User scenarios and hints

- New idea we just started looking into.
- User scenarios: trying to follow a path of a student writing code. Error messages should guide them in the right direction.
- Add hints: sample code snippets and suggestions for what could be causing a problem: "could you be passing parameters in a wrong order"?

Error messages processing

- Minimal interference with IDE or leiningen.
- Need to be consistent with error messages from compiler, run-time system, REPL, testing, other ways of running code?
- Looked into: integrating with IDE (LightTable), leiningen plugin.
- Current plan for catching error messages: `nREPL` middleware.

Leiningen plugin (work in progress)

- Need to load functions with preconditions (and perhaps more) into student code.
- Need to catch errors from various ways of running code and be consistent with processing it.

Integrating with IDEs (work in progress)

- LightTable and Nightcode look promising.
- We may want to display hints: plugins.
- Need to add beginner-friendly project management to IDEs (no command line!)

Future work

- Finish the technical setup for at least one IDE.
- Usability studies: try it on actual students!
- Experiment with hints.
- Modify based on feedback.
- Develop lecture notes (possible future textbook) that focus on CS problem solving while using Clojure.

Many thanks to:

- UMM students who started this project: Brian Goslinga, Stephen Adams, Joe Einertson.
- UMM students who worked on it in 2013/15: Max Magnuson, Paul Schliep, Henry Fellows, Emma Sax, Aaron Lemmon.
- My colleagues and friends: Jon Anthony, Michael Bukatin, Simon Hawkin, Nic McPhee.
- MN Clojure group and Boston Clojure meetup.

Where to find us

- http://cda.morris.umn.edu/ elenam/#clojure
- https://github.com/Clojure-Intro-Course/clojure-intro-class
- https://github.com/elenam