St. Louis Clojure

Clojure Schemata and Generators

Christopher Mark Gore

cgore.com

Tuesday, August 25, AD 2015

Clojure at Climate: We're Hiring!

Add Stuff to Your `project.clj`

```
1  (defproject your-project "1.2.3"
2    :description "It's␣like␣totally␣awesome␣and␣stuff"
3    ;; ...
4    :dependencies [;;...
5                   [prismatic/schema "0.4.3"]
6                   [org.clojure/test.check "0.7.0"]
7                   ;; ...
8                   ]
9    ;; ...
10   )
```

Prismatic Schema

Schemata are sort of like types, but only as strict as you want them to be at that specific moment, so no type hell.

```
1 (ns schema-stuff
2   (:require [schema.core :as s]))
3
4 (s/validate s/Num 42)
5 (s/validate s/Str "howza")
6 (s/validate s/Keyword :hey)
```

Clojure `test.check` and Generators

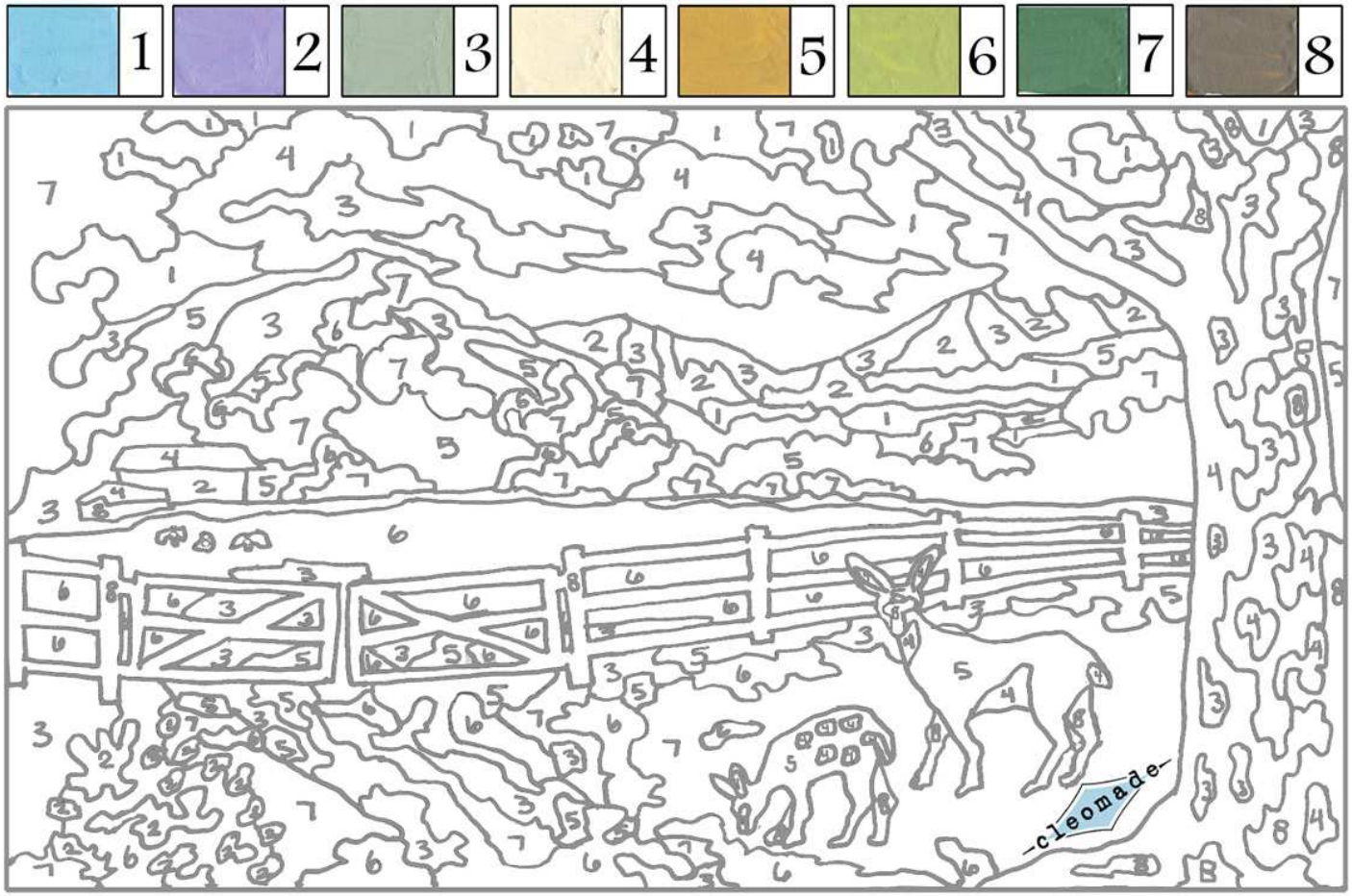
Generators make random examples according to a definition. It's a great way to make test data without brittle hand-rolled examples.

```
1 (ns gen-stuff
2   (:require [clojure.test.check :as tc]
3             [clojure.test.check.generators :as gen]
4             [clojure.test.check.properties :as prop]
5
6 (gen/sample gen/int)
7 ;; => (0 1 -1 0 -1 4 4 2 7 1)
8 (gen/sample gen/int 20)
9 ;; => (0 1 1 0 2 -4 0 5 -7 -8 4 5 3 11 -9 -4 6 -5 -3 0)
```

Schemata + Generators = Awesome!

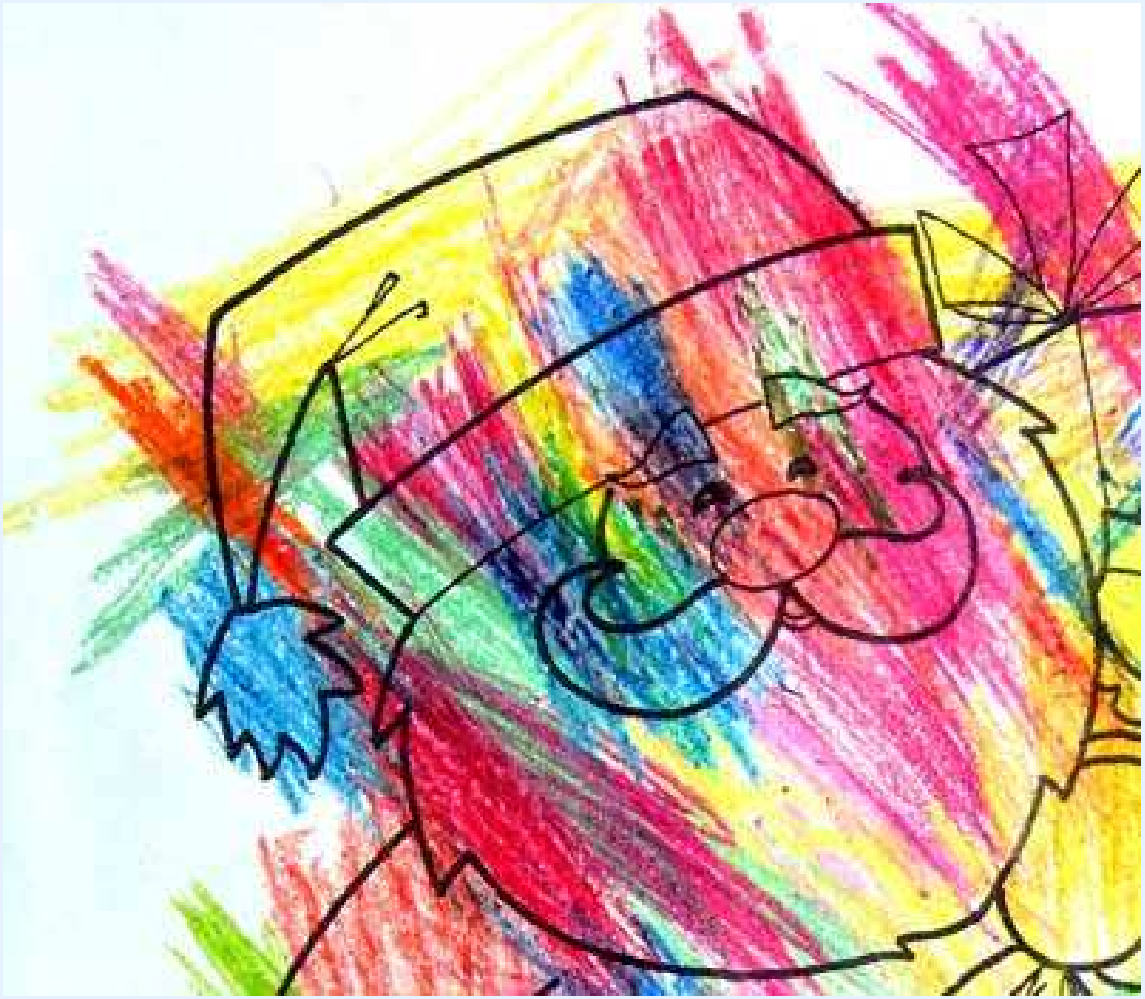
- Schemata to validate function input
  - Definitely in tests.
  - Maybe even in production.
- Generators to fuzz the function in tests.
- Feed the generators into the schemata.
  - Check the generator against the schema.
  - Check the schema with the generator.

Schemata are These



Generators are These

Used Together, We Catch When Our Code Does This



Schema: `validate` versus `check`

The two most important functions for schema checks are `validate` and `check`, the only real difference being that `validate` raises an error and check does not.

```
1 ;; A schema for integers.
2 (s/validate s/Int 42) ; passes, returns 42
3 (try (s/validate s/Int "nope") ; fails, throws an error.
4      (catch Exception e))
5
6 (s/check s/Int 42) ; => nil
7 (s/check s/Int "nope") ; => (not (integer? "nope"))
```

Questions?