What Program Do?

A beginner's guide to Property-Based Testing

What It Is?

"Property-based tests make statements about the output of your code based on the input, and these statements are verified for many different possible inputs."
- Jessica Kerr (@jessitron)

Why Use?

- More code coverage, fewer lines of test code
- Promotes more in-depth understanding
- Aids better requirement/functionality mapping

Replace Unit Tests?

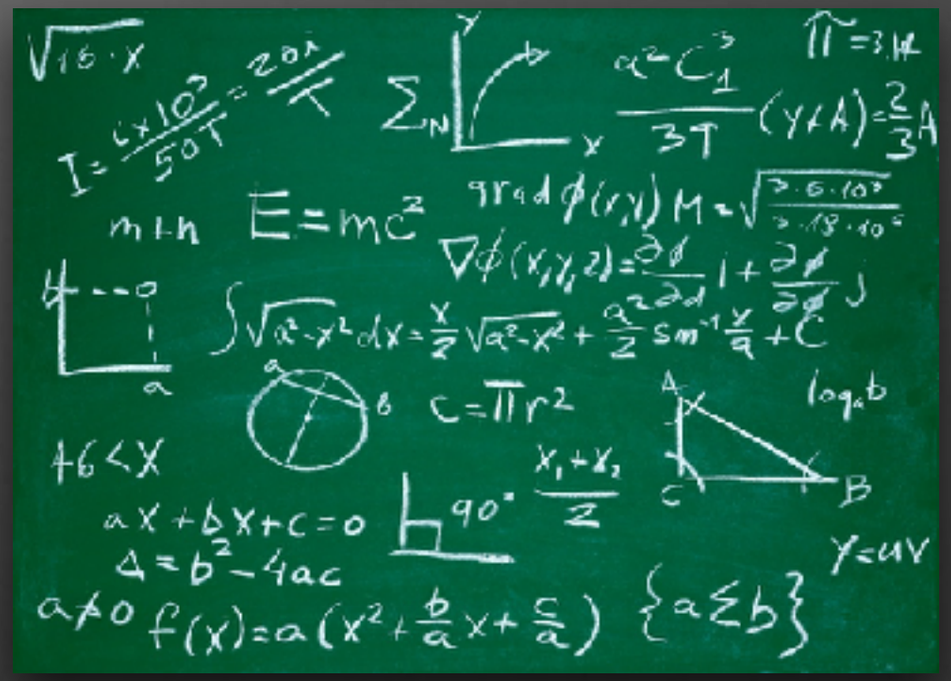
- Unit tests are example-based
  - Expected successes (sin(0) = 0)
  - Expected errors (HTTP POST to ‘/login’ w/ no auth returns 401)
  - Great for early in TDD cycle, but eventually become regression/passive tests
- Property-based tests are generative
  - Expected domain transformations (-1 ≤ sin(x) ≤ 1)
  - Can find unexpected errors (sin(NaN)? I didn’t think about that…)
  - Great for polishing requirements, and are always active tests
- Unexpected errors become unit tests for validating fixes

“Don’t write tests. Generate them!”

–John Hughes, Testing the Hard Stuff and Staying Sane
https://www.youtube.com/watch?v=zi0rHwfiX1Q

What Properties Common?

- Associative — a + (b + c) = (a + b) + c
- Commutative — a + b = b + a
- Distributive — a(b + c) = ab + ac
- Idempotent — a = a * 1 = a * 1 * 1



“Math is easy; design is hard.”

- Jeffrey Veen

How Use in Not Math?

- Associative
  - `users.Sort().Filter(x => !x.IsAdmin) = (users.Filter(x => !x.IsAdmin)).Sort()`
- Commutative
  - `image.flipX().flipY() = image.flipY().flipX()`
- Distributive
  - `title.ToUpper() + author.ToUpper() = (title + author).ToUpper()`
- Idempotent
  - `title.Trim() = title.Trim().Trim()`
  - `list.Sort() = list.Sort().Sort()`

Properties Not Look Mathy?

- Bilbo Testing (aka, There And Back Again)
  - `list = list.Reverse().Reverse()`
  - `obj = JSON.parse(JSON.stringify(obj))`
- No Unexpected Change
  - `list.Length = list.Sort().Length`
- Should Not Crash
- Hard to Prove, Easy to Verify
  - Sorting (`list.Pairwise().All((x, y) => x ≤ y)`)

Used by People?

- Volvo Uses QuviQ QuickCheck for its Embedded System
  - Property-Based Testing Quite Literally Saves Lives
- Clojure 1.5 Transient ↔ Persistent Structures
  - Hash Code Equality Issue (http://dev.clojure.org/jira/browse/CLJ-1285)
  - Example-Based Coding for Transient to/from Persistent Data Structures Code Did Not Catch the Error
  - Property-Based Testing Reliably Finds the Issue
- FASTER ANT — Packet Fingerprinting
  - Generated 100 million random TCP/UDP packets with each push to CI server to ensure no false matches

Other Things for Help?

- Shrinking — specifying the minimum input required to fail the property/ hypothesis
- Race Conditions — specifying arbitrary actions run in parallel to detect race conditions that are traditionally hard to replicate in unit tests
- Custom generators — Create input domain constraints, such as
  - Only printable characters
  - Floating point values between 0 and 1
- Test Oracle — when refactoring an old implementation, use PBT to verify new implementation yields same results as original

Info Links?

- Hypothesis, Quick Check in Every Language: https://hypothesis.works/articles/quickcheck-in-every-language/
- John Hughes, Testing the Hard Stuff and Staying Sane: https://www.youtube.com/watch?v=zi0rHwfiX1Q
- Fred Herbert, PropEr Testing: http://propertesting.com
- Jessica Kerr, Property-based Testing: What Is It?: http://blog.jessitron.com/2013/04/property-based-testing-what-is-it.html
- Scott Wlaschin, Intro to Property-Based Testing: https://fsharpforfunandprofit.com/posts/property-based-testing/
- GitHub repo for demos: https://github.com/jdsteinhauser/pbt-intro

Thanks! Questions?