Property Based Testing:
Let the Computer Make Tests for You

Peter Schuck
@spinningtopsofdoom / @bendyworks

Unit Testing pain points

- Linear growth with code size
- Snowflake test inputs
- Zombie tests

Eris

Property Test Library for PHP

https://github.com/giorgiosironi/eris

Source code for all examples is at

https://github.com/spinningtopsofdoom/madison-php-2016

Overview from spaaaaace of Property based testing

Typical Unit Tests

```
add2(3, 7) === 10;
sort(["c", "a", "b"]) === ["a", "b", "c"];
```

Property Based Tests

```
add2($a, $b) === add2($b, $a);
sort($items) === sort(sort($items));
```

Properties are tested via generated inputs

```
$add2_generator = Generator\vector(2, Generator\int());
$sort_generator = Generator\seq(Generator\string());
```

Let’s go through a strawman example

Latinify function

Transforms the word "cat" to "Felinus"

```
latinify("cat") === "Felinus";
```

Basic unit tests for latinify

```
latinify("no felines here") === "no felines here";
latinify("cat") === "Felinus";
latinify("Where is the cat?") === "Where is the Felinus?";
```

Totally valid function

```
function latinify($plain_string) {
  if ($plain_string === "cat") {
    return "Felinus";
  } elseif ($plain_string === "Where is the cat?") {
    return "Where is the Felinus?";
  }
  return $plain_string;
}
```

Lets add more tests

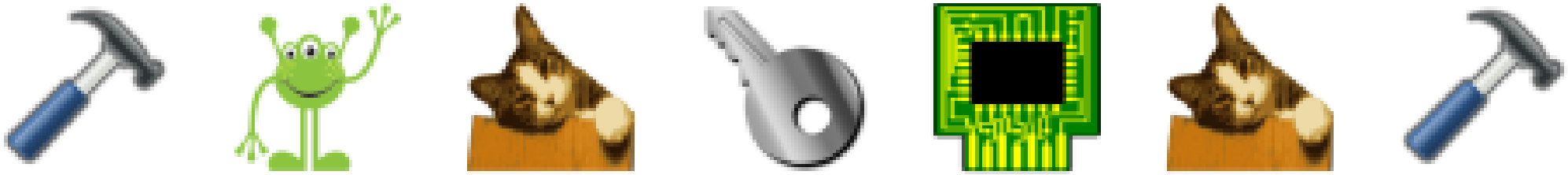
```
latinify("fat cat") === "fat Felinus";
latinify("cat with a hat") === "Felinus with a hat";
```

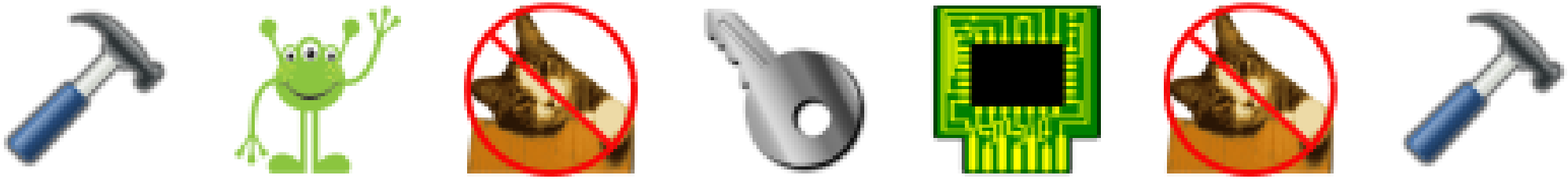
Still a valid function

```
function latinify($plain_string) {
  if ($plain_string === "cat") {
    return "Felinus";
  } elseif ($plain_string === "Where is the cat?") {
    return "Where is the Felinus?";
  } elseif ($plain_string === "fat cat") {
    return "fat Felinus";
  } elseif ($plain_string === "cat with a hat") {
    return "Felinus with a hat";
  }
  return $plain_string;
}
```

Now lets try property based testing

The output string should contain no "cat"s

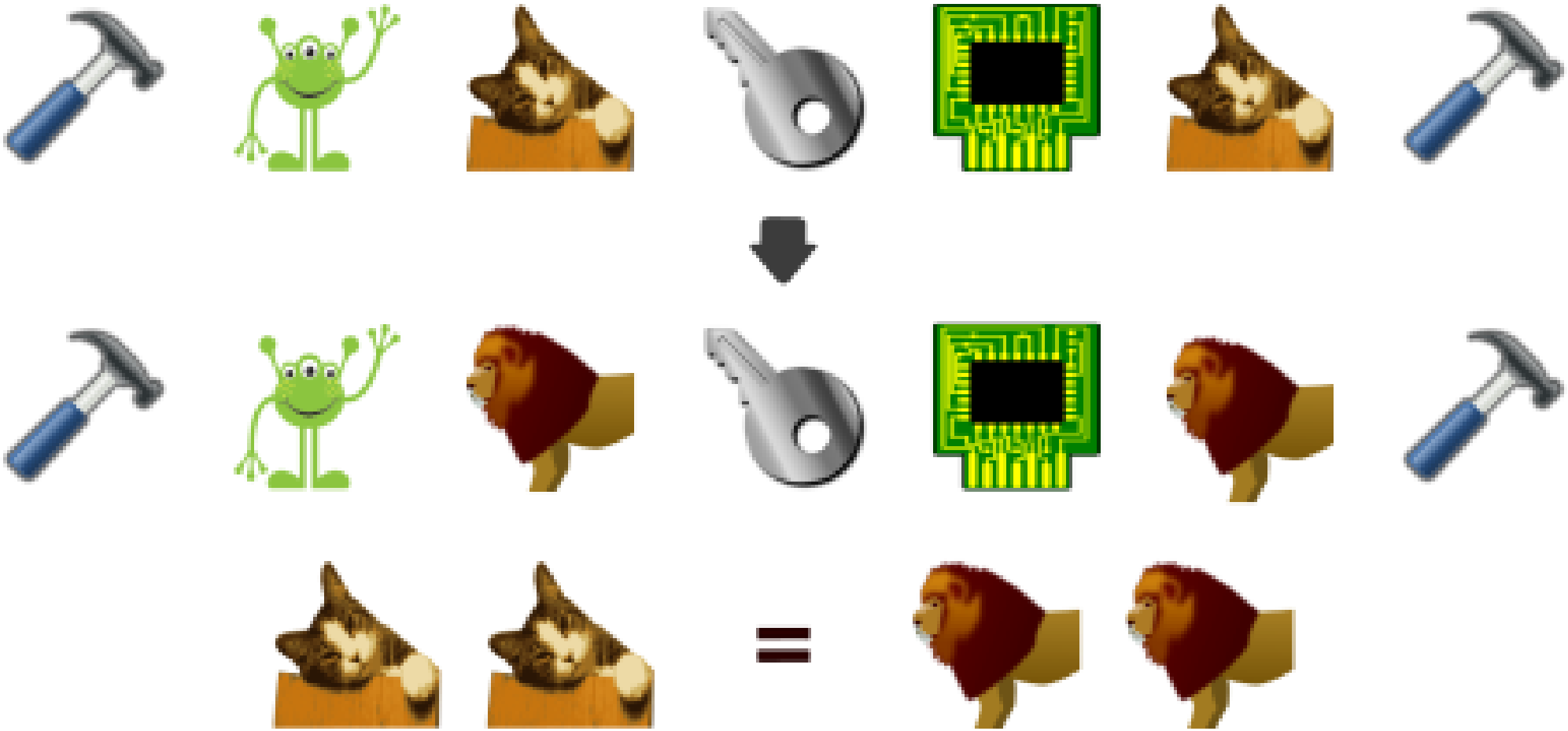




Our function is even easier to fake

```
function latinify($plain_string) {
  return "No Felinus here :p";
}
```

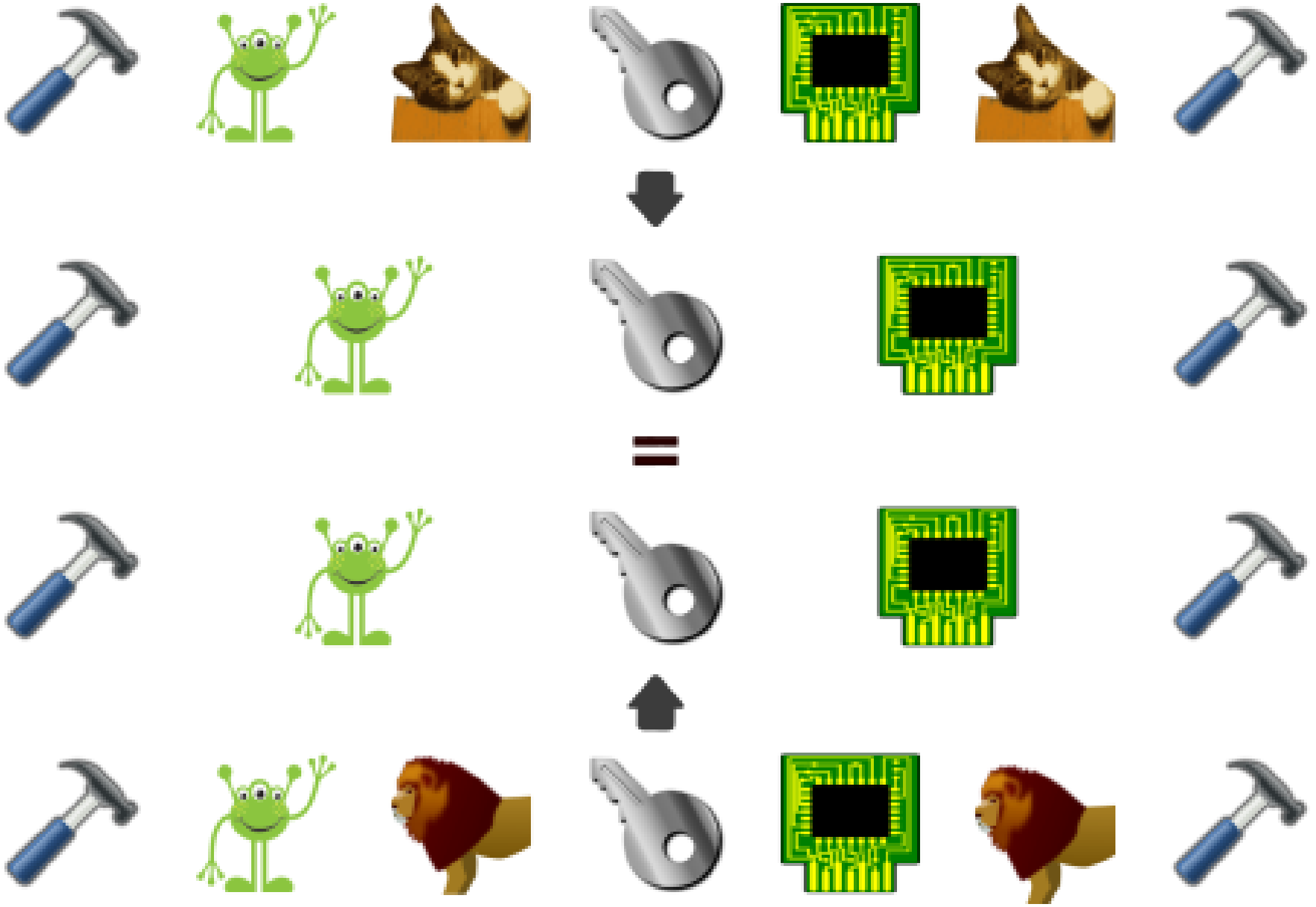
The output string has as many "Felinus" as the input has "cat"s



Closer but still wrong

```
function latinify($plain_string) {
  return str_replace("cat", "Felinus behind us", $plain_string);
}
```

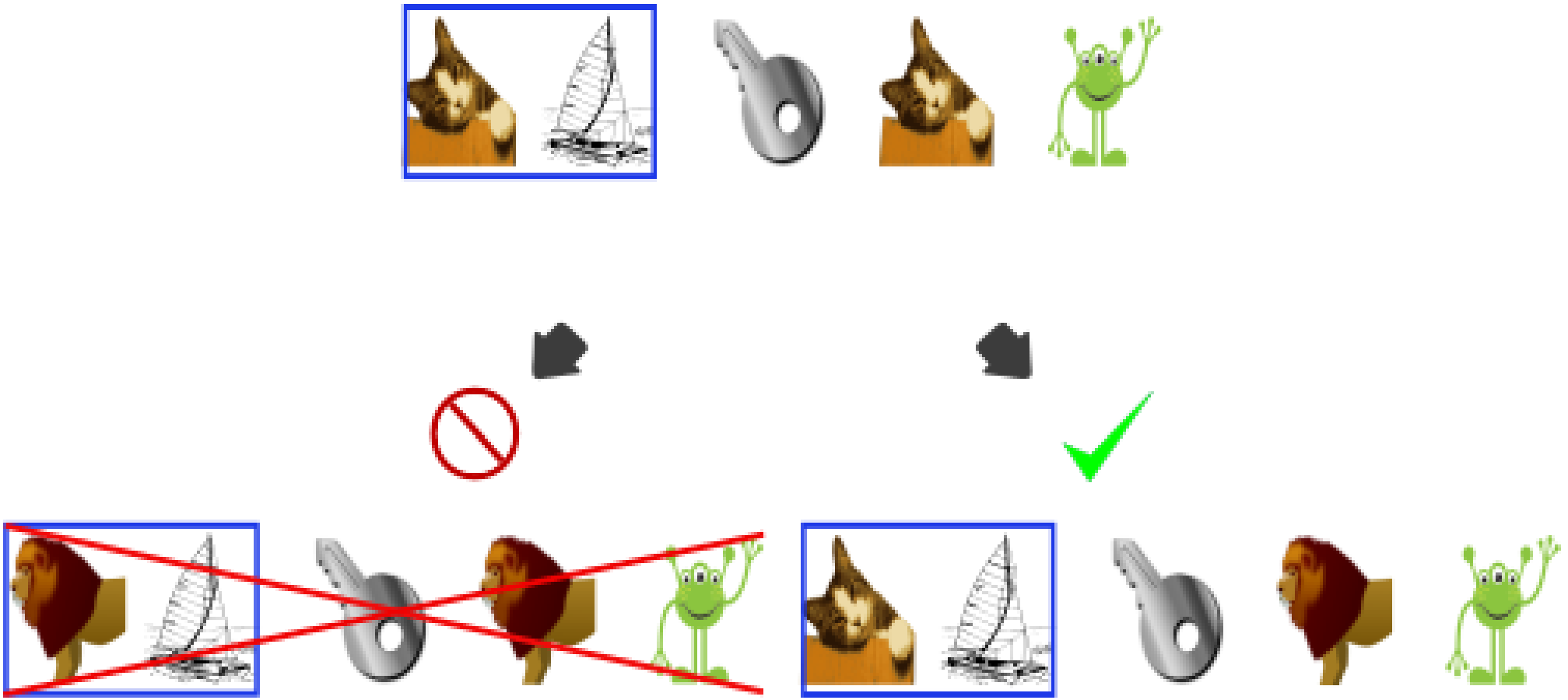
Removing 'Felinus' and 'cat' should get the same string



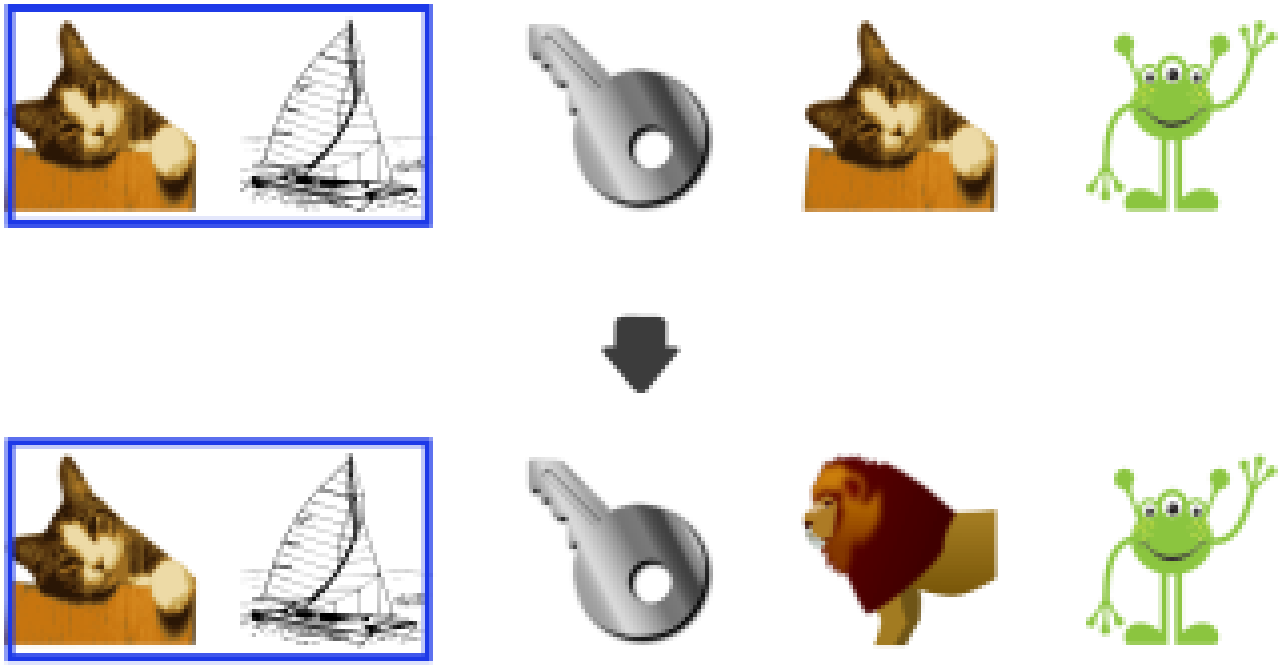
Finally forced to create a reasonable function

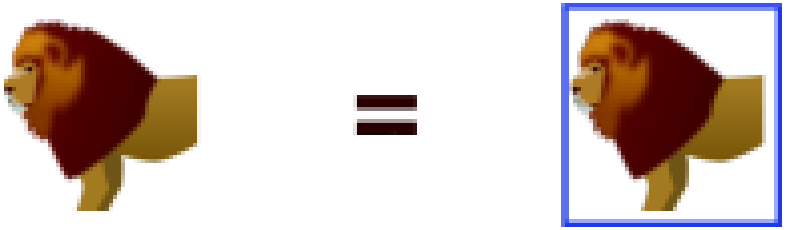
```
function latinify($plain_string) {
  return str_replace("cat", "Felinus", $plain_string);
}
```

What about "Hepcat on a catamaran"?



Number of "Felinus" words matches the number of "Felinus" strings





Final function only replaces "cat" when it is a word

```
function latinify($plain_string) {
  return preg_replace('/\bcat\b/', "Felinus", $plain_string);
}
```

Finding Properties to test

1) Write some unit tests
2) Construct bad function / method that passes unit tests
3) Find property to fail function / method
4) Refactor bad function / method to slip by property test
5) Repeat 3 – 4 until satisfied with covage

Common Properties

- Action gives same result run 1 or N times
  - Sorting an array
- Reversible operations
  - Converting PHP <=> JSON
- Actions can occur in any order
  - Adding items to a shopping cart

Talks listing common properties

An introduction to property based testing
by Scott Wlaschin

Real World Property Based Testing
By Charles O'Farrell

Property Testing has Huge Costs

- Thinking of properties to test 1 – 2 orders of magnitude slower
- Generators need to be made
- Tests take much longer to run

Great power to weight ratio

- A unit test catches one bug
- A property test catches a whole class of bugs
- Lower marginal cost when code changes happen

Unit Tests are still very useful

Unit tests uses

- Great during exploration / prototype phase
- Fast Failure
- Documenting edge cases

Eris Generators

Features and examples

Generators provide random inputs for tests

- Composable, you can build a bigger generator from smaller ones
- Repeatable, a failing input can be repeated
- Shrinkable, The killer feature of property test libraries

What is shrinking?

Find the misspelled word

Deductibles are typically used to deter the large number of claims that a consumer can be reasonibly expected to bear the cost of. By restricting its coverage to events that are significant enough to incur large costs, the insurance firm expects to pay out slightly smaller amounts much less frequently, incurring much higher savings.

Deductibles are typically used to deter the large number of claims that a consumer can be reasonibly expected to bear the cost of.

that a consumer can be reasonibly expected to bear the cost of.

that a consumer can be reasonibly

Shrinking takes a failing input and finds the smallest portion of it that fails the test

reasonibly

Basic Eris Generators

```
Generator\nat(); // 3, 56, 1, 32
Generator\string(); // "K", "g,", "jGHr38i"
Generator\float(); // 3.87, 0.0, 1.7
Generator\bool(); // true, true, false
```

Eris composite generator example

Collection of 10 temperature readings

Each reading looks like

```
['scale' => 'F', 'degrees' => 50]
```

Basic Setup

```
$scale = Generator\elements(['C', 'F', 'K']);
$degrees = Generator\choose(0, 100));
$reading = Generator\associative(
  ['scale' => $scale,
   'degrees' => $degrees]);
$measurements = Generator\vector(10, $reading);
```

More realistic scale distribution

60% F - 30% C - 10% K

Frequency: Changes the probability that generator will be chosen from a collection of generators

```
$fahrenhiet = Generator\constant('F');
$celsius = Generator\constant('C');
$kelvin = Generator\constant('K');
$dist_scale = Generator\frequency(
  [6, $fahrenhiet],
  [3, $celsius],
  [1, $kelvin]);
```

Temperatures should match scale

100 C or 0 K = :(

Bind: Generator => Output => Function => Generator

```
$realistic_reading = Generator\bind(
  $dist_scale,
  function($scale) {
    $degrees = ['F' => Generator\choose(0, 100),
                'C' => Generator\choose(0, 32),
                'K' => Generator\choose(280, 310)];
    return Generator\associative(['scale' => $scale,
      'degrees' => $degrees[$scale]]);
  }
);
```

More accurate readings

85 degrees => 85.94 degrees

Map: Generator => Output => Function => Modified Output

```
$accurate_reading = Generator\map(
  function($gen_data) {
    list($precision, $reading) = $gen_data;
    $degrees = round($reading['degrees'] + $precision, 2);
    $reading['degrees'] = $degrees; return $reading;
  },
  Generator\tuple(Generator\float(), $realistic_reading));
```

Have at least one Kelvin Reading

[C, C, C, C, C, F, F, F, F, F] ☒

[C, C, C, C, C, K, F, F, F, F] ☑

SuchThat: Generator => Invalid Output => Check => Invalid Output => Check => Valid Output

```
$measurements = Generator\vector(10, $accurate_reading);
$one_k_measurements = Generator\suchThat(
  function ($measurements) {
    return array_reduce(
      $measurements,
      function($one_k, $reading) {
        return $one_k || ($reading['scale'] === 'K');
      }, false);
  },
  $measurements);
```

Generators

- Force mapping out problem domain
- No more copy pasta of hand rolled inputs
- Easily create sample data for new functionality

So far we've tested functions and created generators.

How can you property test objects?

Friendship tracker with methods

- `addFriendship('Bob','Alice')`
  - Adds a new friendship between 'Bob' and 'Alice'
- `removeFriendship('Bob','Alice')`
  - Removes the friendship between 'Bob' and 'Alice'
- `friends('Alice','Bob')`
  - Checks if 'Alice' and 'Bob' have a friendship
- `getFriends('Alice')`
  - Get all of 'Alice's friends
- `getPeople()`
  - Gets the people in our database ('Alice' and 'Bob')

Typical Usage

```
$friends = new Friendships();
$friends->addFriendship('Alice', 'Bob');
$friends->addFriendship('Alice', 'Carol');
$friends->addFriendship('Dan', 'Bob');
$friends->removeFriendship('Alice', 'Bob');
```

We can represent this in data

```
$operations = [['addFriendship', 'Alice', 'Bob'],
  ['addFriendship', 'Alice', 'Carol'],
  ['addFriendship', 'Dan', 'Bob'],
  ['removeFriendship', 'Alice', 'Bob']];
```

Similar to a todo list

The actions and information on the list are not the actual actions

Todos

- Pick up milk on the way home
- Pet the nice cat on daily constitutional
- Prove that P ≠ NP

No self friending

Symmetrical Friendship

⇐

⇒

Create a new Friendship tracker

```
$fresh_friendship = Generator\map(
  function($friendship_map) {
    return new Friendships();
  },
  Generator\associative([]));
```

Friend friendship operation generator

```
$methods = Generator\elements([
  'addFriendship',
  'removeFriendship']);
$operation = Generator\tuple(
  $methods,
  Generator\elements($people),
  Generator\elements($people));
$operations = Generator\seq($operation);
```

Create the Friend Tracker with operations that modified it's state

```
$modified_friendship = Generator\bind(
  $operations,
  function($operations) use ($fresh_friendship) {
    return Generator\map(
      function($friendship) use ($operations) {
        $new_friendship = apply_operations($friendship,
                                           $operations);
        return [$new_friendship, $operations];
      },
      $fresh_friendship);
  }
);
```

Fresh Object Generator

+

Array of operations to perform

=>

Modified Object with trace of how it got to it’s current state

Current Eris sharp edges
(as of 01-Oct-2016)

Shrinking is currently very poor (linear time)

Hypothesis (A Python Property Testing Library) has done a lot of work to making shrinking logarithmic time

Highly coupled with PHPUnit

Can't play around with generators without arcane boilerplate

```
require 'vendor/autoload.php';
Eris\TestTrait::erisSetupBeforeClass();
use Eris\Generator;

function sample($generator, $times, $seed = 10) {
  mt_srand($seed);
  return Eris\Sample::of($generator, 'mt_rand')
           ->repeat($times)
           ->collected();
}
```

Uneven Documentation

Great for simple cases, complex use cases sorely lacking

Property Testing eases testing pain points

- Cut down on code size and churn
- Compliments Unit tests
- Generators have multiple uses outside of tests

Fin

Questions?