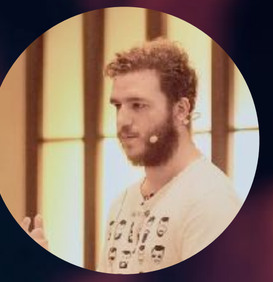

PROPERTY-BASED TESTING
IS A MINDSET

@whatyouhide 


weedmaps®

weedmaps.com

weedmaps

Search for dispensaries, deliveries & brands

Irvine, CA

LOG IN

SIGN UP

HOME | MAPS | BRANDS | DEALS | LEARN | ORDER ONLINE BETA

3 BUSINESS TYPES

FILTER

MARIJUANA LISTINGS in Santa Ana, California

Church of the Holy Grail - Newp...
★★★★★ 5.0 BY 1837 REVIEWS
NEWPORT BEACH, CA
OPEN NOW — 514 MENU ITEMS

ShowGrow Santa Ana
★★★★★ 4.7 BY 859 REVIEWS
SANTA ANA, CA
OPEN NOW — 449 MENU ITEMS

Unity of ONAC
★★★★★ 4.7 BY 932 REVIEWS
IRVINE, CA
OPEN NOW — 262 MENU ITEMS

OC Compassionate Care - Cost...
★★★★★ 4.9 BY 88 REVIEWS
COSTA MESA, CA
OPEN NOW — 157 MENU ITEMS

Organix Delivery - Tustin
★★★★★ 4.8 BY 233 REVIEWS
TUSTIN, CA

HOME > SANTA ANA LISTINGS > SANTA ANA LISTINGS

Walnut
Northwood
El Camino Real
Westpark
Irvine
Woodbury
Edinger Ave
Valencia Ave
Red Hill Ave
E Warner Ave
Barranca Pkwy
Tustin Ranch Rd
McGaw Ave
Alton Pkwy
Armstrong Ave
Von Karman Ave
Jamboree Rd
Harvard Ave
Culver Dr
W Yale Loop
Main St
San Diego Fwy
Paseo Westpark
Teller Ave
Michelson Dr
University Dr
San Joaquin Marsh Reserve
Ridgeline Dr

7 Grams for $50 Private Reserve!

Whole Store

KUSHAGRAM

https://weedmaps.com/careers


T E S T I N G

why do we test?

no tests 😟

yes tests 😔

unit tests

```
test "sorting" do
  assert sort([]) == []
  assert sort([1, 2, 3]) == [1, 2, 3]
  assert sort([2, 1, 3]) == [1, 2, 3]
end
```

example-based

Input	Output
`[]`	`[]`
`[1, 2, 3]`	`[1, 2, 3]`
`[2, 1, 3]`	`[1, 2, 3]`

table-based

unit tests

🙂, but also 😢


PROPERTIES

hard to write 😟, but...


valid **inputs**
properties of **output**
testing framework

github.com/whatyouhide/**stream_data**

example time:

sorting lists

```
test "sorting" do
  assert sort([]) == []
  assert sort([1, 2, 3]) == [1, 2, 3]
  assert sort([2, 1, 3]) == [1, 2, 3]
end
```

lists of **integers**

it's a list

has the same elements

it's ordered

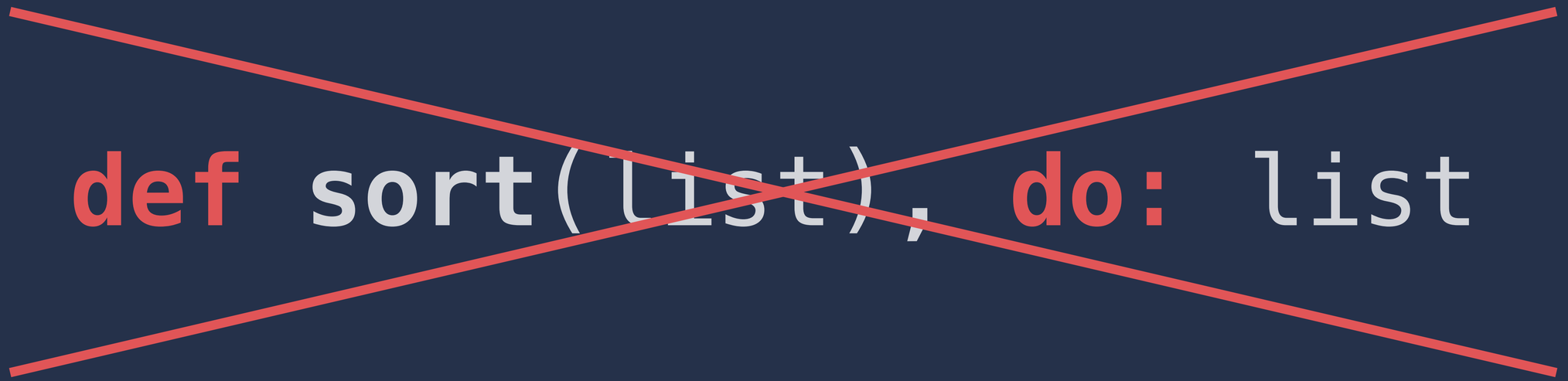
```
check all list <- list_of(int()) do
  sorted = sort(list)

  assert is_list(sorted)
  assert same_elements?(list, sorted)
  assert ordered?(sorted)
end
```

```
check all list <- list_of(int()) do
  sorted = sort(list)

  assert is_list(sorted)
  assert same_elements?(list, sorted)
  assert ordered?(sorted)
end
```

```
check all list <- list_of(int()) do
  sorted = sort(list)

  assert is_list(sorted)
  assert same_elements?(list, sorted)
  assert ordered?(sorted)
end
```

```
check all list <- list_of(int()) do
  sorted = sort(list)

  assert is_list(sorted)
  assert same_elements?(list, sorted)
  assert ordered?(sorted)
end
```

```
check all list <- list_of(int()) do
  sorted = sort(list)

  assert is_list(sorted)
  assert same_elements?(list, sorted)
  assert ordered?(sorted)
end
```


def sort(list), do: list

[32, 2, 44, -12]

[1, 0]


G E N E R A T O R S

```
iex> Enum.take(integer(), 4)
[1, 0, -3, 1]
```

Composability

```
number = one_of([integer(), float()])
```

```
StreamData.map(integer(), &abs/1)
```

Example

```
def string(:ascii) do
  integer(?\s..?~)
  |> list_of()
  |> map(&List.to_string/1)
end
```

Keep shrinkability

```
constant(term)
```

+

```
bind_filter(gen, fun)
```

constant always generates a term

```
iex> Enum.take(constant(:foo), 4)
[:foo, :foo, :foo, :foo]
```

bind_filter (possibly) creates generators from generated terms

```
bind_filter(integer(), fn i ->
  if i < 0 do
    :skip
  else
    gen = map(list_of(integer()), &(&1 + i))
    {:cont, gen}
  end
end)
```


PATTERNS

circular code

```
decode(encode(term)) == term
```

JSON encoding

```
property "unicode escaping" do
  check all string <- string(:printable) do
    encoded = encode(string, escape: :unicode)
    assert decode(encoded) == string
  end
end
```

oracle model

```
my_code() == oracle_code()
```

older system

less performant implementation

```
property "gives same results as Erlang impl" do
  check all bin <- binary() do
    assert Huffman.encode(bin) ==
             :huffman.encode(bin)
  end
end
```

smoke tests

https://www.youtube.com/watch?v=jvwfDdgg93E

API: 200, 201, 400, 404

https://www.youtube.com/watch?v=jvwfDdgg93E

```
property "only expected codes are returned" do
  check all request <- request() do
    response = HTTP.perform(request)
    assert response.status in [200, 201, 400, 404]
  end
end
```

https://www.youtube.com/watch?v=jvwfDdgg93E

locally  , **CI** 

```elixir
if ci?() do
  config :stream_data, max_runs: 500
else
  config :stream_data, max_runs: 25
end
```

unit + properties

```elixir
property "String.contains?/2" do
  check all left <- string(),
            right <- string() do
    assert String.contains?(left <> right, left)
    assert String.contains?(left <> right, right)
  end
end
```



```elixir
test "String.contains?/2" do
  assert String.contains?("foobar", "foo")
  assert String.contains?("foobar", "bar")
  assert String.contains?("foobar", "ob")
end
```


STATEFUL
TESTING

model

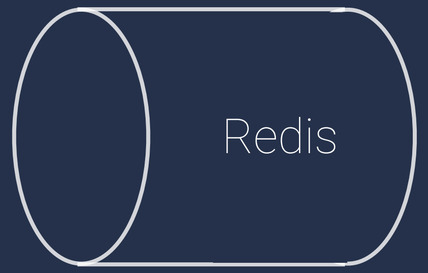
valid commands

model: state + state transformations

commands: calls + preconditions

getting/setting keys
in Redis

system



model

%{}

commands

- `get(key)`
- `set(key, value)`

```
def get(model, key),
  do: Map.get(model, key)


def set(model, key, value),
  do: Map.put(model, key, value)
```

```
keys = Map.keys(model)

one_of([
  command(:get, [one_of(keys)]),
  command(:set, [binary(), binary()]),
  command(:set, [one_of(keys), binary()])
])
```

get

```
{:ok, result} = Redix.command!(conn, ["GET", key])
assert Map.fetch(model, key) == {:ok, result}
```

set_existing

```
Redix.command!(conn, ["SET", key, value])
Map.replace!(model, key, value)
```

LevelDB

17 (seventeen) calls

33 (thirty three) calls


R E S E A R C H

trees are hard

```elixir
defmodule Tree do
  def tree() do
    StreamData.tree(:leaf, fn leaf -> {leaf, leaf} end)
  end

  def size({l, r}), do: 1 + size(l) + size(r)
  def size(_leaf), do: 1

  def depth({l, r}), do: 1 + max(depth(l), depth(r))
  def depth(_leaf), do: 1
end
```

Generation size: 10
Avg size: 4.9
Avg max depth: 2.473

Generation size: 100
Avg size: 10.892
Avg max depth: 3.466

Generation size: 1000
Avg size: 22.732
Avg max depth: 4.507

stream_data

+

dialyzer

Google Summer of Code

Generators from type

```
@type timeout() :: :infinity | non_neg_integer()
```

```
from_type(timeout())
```

one_of([:infinity, map(integer(), &abs/1)])

Automatic typespec property checking

```elixir
@spec my_fun(timeout()) :: :ok | :error
```

```elixir
check all timeout <- from_type(timeout()) do
  assert my_fun(timeout) in [:ok, :error]
end
```


C O N C L U S I O N

find obscure bugs

reduce to minimal failing input

find specification errors

cover vast input space


v1.7

use stream_data

use property-based testing


@whatyouhide
github.com/whatyouhide/stream_data