use Test::Base;

Tatsuhiko Miyagawa
miyagawa@gmail.com

Six Apart, Ltd. / Shibuya Perl Mongers
Shibuya.pm Tech Talks #7

Test::Base
とは

データドリブン
テストベースクラス

by Ingy döt Net

use Test::More?

use Test::Base!

Test::More と互換のあるインターフェース

Test::More compatible

```
use Test::More;
plan tests => 2;
sub camelize {
    local $_ = shift;
    s/_(¥w)/uc($1)/eg;
    $_;
}

is camelize("foo_bar"), "fooBar";
is camelize("foo_bar_baz"), "fooBarBaz";
```

Test::More compatible

```
use Test::Base;
plan tests => 2;
sub camelize {
    local $_ = shift;
    s/_(¥w)/uc($1)/eg;
    $_;
}

is camelize("foo_bar"), "fooBar";
is camelize("foo_bar_baz"), "fooBarBaz";
```

Test::Base-y code

```
use Test::Base;
sub camelize {
    local $_ = shift;
    s/_(¥w)/uc($1)/eg;
    $_;
}
filters { input => 'camelize' }
__DATA__
===
--- input: foo_bar
--- expected: fooBar
===
--- input: foo_bar_baz
--- expected: fooBarBaz
```

Test::Base code

```
# 01-json.t
use Test::Base;
use JSON::Syck;
sub json { JSON::Syck::Dump($_[0]) }
run_is 'input' => 'expected';
__END__
=== Simple JSON Test
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}
```

TAP compatible

```
% prove -l 01-json.t
01-json....ok
All tests successful.
Files=1, Tests=1,  0 wallclock secs ( 0.10 cusr +
   0.02 csys =  0.12 CPU)
```

use Test::Base

```
# 01-json.t
use Test::Base;
use JSON::Syck;
sub json { JSON::Syck::Dump($_[0]) }
run_is 'input' => 'expected';
__END__
=== Simple JSON Test
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}
```

Block names

```
# 01-json.t
use Test::Base;
use JSON::Syck;
sub json { JSON::Syck::Dump($_[0]) }
run_is 'input' => 'expected';
__END__
=== Simple JSON Test
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}
```

Filters

```
# 01-json.t
use Test::Base;
use JSON::Syck;
sub json { JSON::Syck::Dump($_[0]) }
run_is 'input' => 'expected';
__END__
=== Simple JSON Test
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}
```

perldoc Test::Base::Filter

Define Filters

```
# 01-json.t
use Test::Base;
use JSON::Syck;
sub json { JSON::Syck::Dump($_[0]) }
run_is 'input' => 'expected';
__END__
=== Simple JSON Test
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}
```

Filter arguments

```
=== filter args
--- input lines array tail=2 Join
foo
Bar
baz
--- expected chomp regexp=xis
bar.*baz
```

filters

```
__DATA__
=== test 1
--- input foo bar baz
xxx
--- expected quox
yyy

=== test 2
--- input foo bar baz
xxx
--- expected quox
yyy

=== test 3
--- input foo bar baz
xxx
--- expected quox
yyy
```

filters

```
filters { input => [ 'foo', 'bar', 'baz' ], expected => 'quox' };
__DATA__
=== test 1
--- input
xxx
--- expected
yyy

=== test 2
--- input
xxx
--- expected
yyy

=== test 3
--- input
xxx
--- expected
yyy
```

run_is

```
# 01-json.t
use Test::Base;
use JSON::Syck;
sub json { JSON::Syck::Dump($_[0]) }
run_is 'input' => 'expected';
__END__
===
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}
```

Auto-diff

```
01-json....NOK 1
#   Failed test '
# @@ -1,2 +1,2 @@
# -{"foo":"bar"}
# +{ foo => "bar" }
#  xxx
# '
```

Requires Algorithm::Diff
Turn off with no_diff()

run_is

run_like

run_unlike

run_is_deeply

run_compare

run_is default blocks

```
# 01-json.t
use Test::Base;
use JSON::Syck;
sub json { JSON::Syck::Dump($_[0]) }
run_is;
__END__
===
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}
```

run_is default blocks

```
# 01-json.t
use Test::Base;
use JSON::Syck;
sub json { JSON::Syck::Dump($_[0]) }
run_is;
__END__
===
--- xxx eval json
{ foo => "bar" }
--- yyy chomp
{"foo":"bar"}
```

run_compare

```
# 01-json.t
use Test::Base;
use JSON::Syck;
sub json { JSON::Syck::Dump($_[0]) }
run_compare;
__END__
===
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}

===
--- input eval json
{ foo => [ "bar", "baz" ] }
--- expected regexp
bar.*baz
```

run_compare is default!

```
# 01-json.t
use Test::Base;
use JSON::Syck;
sub json { JSON::Syck::Dump($_[0]) }
__END__
===
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}

===
--- input eval json
{ foo => [ "bar", "baz" ] }
--- expected regexp
bar.*baz
```

run

```
# 01-json.t
use Test::Base;
use JSON::Syck;
plan tests => 1 * blocks;
run {
   my $block = shift;
   is JSON::Syck::Dump($block->input), $block->expected, $block->name;
};
__END__
===
--- input eval
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}
```

blocks()

```
# 01-json.t
use Test::Base;
use JSON::Syck;
plan tests => 1 * blocks;
for my $block (blocks) {
   is JSON::Syck::Dump($block->input), $block->expected, $block->name;
}
__END__
===
--- input eval
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}
```

Block specific

```
use Test::Base;
run_compare;
__END__
===
--- ONLY
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}

===
--- input eval json
{ foo => [ "bar", "baz" ] }
--- expected regexp
bar.*baz
```

Block specific

```
use Test::Base;
run_compare;
__END__
===
--- SKIP
--- input eval json
{ foo => "bar" }
--- expected chomp
{"foo":"bar"}

===
--- input eval json
{ foo => [ "bar", "baz" ] }
--- expected regexp
bar.*baz
```

Subclassing Test::Base

```
# t/TestJSON.pm
package t::TestJSON;
use Test::Base -Base;
use JSON::Syck;

package t::TestJSON::Filter;
use Test::Base::Filter –base;

sub json { JSON::Syck::Dump($_[0]) }

# t/01-json.t
use t::TestJSON;
__END__
===
--- input eval json
{ foo => "bar" }
--- expected
…
```

In your CPAN modules

```
use inc::Module::Install;
name 'Foo-Bar';
all_from 'lib/Foo/Bar.pm';
use_test_base;
WriteAll;
```

Tips

Spiffy XXX

```
XXX $foo = $bar;
WWW $bar;
YYY $baz;
ZZZ $quox;
```

no chomp filters

```
use Test::Base;
filters { input => 'chomp', expected => 'chomp' }
__END__
===
--- input
foo
--- expected
bar
===
--- input
fooaba
--- expected
bzaahiepa
```

no chomp filters

```
use Test::Base;
__END__
===
--- input: foo
--- expected: bar
===
--- input: fooaba
--- expected: bzaahiepa
```

success tests vs. error tests

```
# t/01-success.t
run_is 'input' => 'expected';

# t/02-failure.t
run {
    my $block = shift;
    eval { … };
    run_like $@, $block->expected;
};
```

Test::Base に向かないテスト

複雑なAPI

オ　ジェ　　状態が変化

Test::Base で
うまく　　できない
＝　　API

Design your API Sufficiently Advanced.

© Damian Conway

References

perldoc Test::Base

http://search.cpan.org/~miyagawa/?R=D

JSAN
Test.Base.js

(PMConnect?)

Refactoring Example

http://yapc.kwiki.org/data-driven-testing/slide10.html