Mamba Documentation

Release 0.9.2

Néstor Salceda

Oct 24, 2018

Contents

Mamba is a *Behaviour-Driven Development* tool for Python developers.

Is heavily influenced from RSpec, Mocha, Jasmine or Ginkgo.

This is the official documentation site for Mamba.

CHAPTER 1

Contents

1.1 Getting Started

1.1.1 Prerequisites

Python 2.7 or higher.

1.1.2 Installation

I recommend to use pipenv for managing your dependencies, thus you can install mamba like any other Python package.

By example:

```
$ pipenv install mamba
```

But you also can use pip:

```
$ pip install mamba
```

1.1.3 Your first example

Write a very simple example that describes your code behaviour:

```
# tennis_spec.py

from mamba import description, context, it
from expects import expect, equal

with description('Tennis') as self:
  with it('starts with 0 - 0 score'):
```

(continues on next page)

(continued from previous page)

```
    rafa_nadal = "Rafa Nadal"
    roger_federer = "Roger Federer"
    game = Game(rafa_nadal, roger_federer)

    expect(game.score()).to(equal((0, 0)))
```

Run the example, and don't forget to watch it fail!

```
$ pipenv run mamba tennis_spec.py

F

1 examples failed of 1 ran in 0.0023 seconds

Failures:

  1) Tennis it starts with 0 - 0 score
    Failure/Error: tennis_spec.py game = Game(rafa_nadal, roger_federer)
        NameError: global name 'Game' is not defined

    File "tennis_spec.py", line 8, in 00000001__it starts with 0 - 0 score--
        game = Game(rafa_nadal, roger_federer)
```

Now write as little code for making it pass.

```
# tennis_spec.py

from mamba import description, context, it
from expects import expect, equal

import tennis

with description('Tennis') as self:
  with it('starts with 0 - 0 score'):
    rafa_nadal = "Rafa Nadal"
    roger_federer = "Roger Federer"
    game = tennis.Game(rafa_nadal, roger_federer)

    expect(game.score()).to(equal((0, 0)))
```

```
# tennis.py

class Game(object):
  def __init__(self, player1, player2):
    pass

  def score(self):
    return (0, 0)
```

Run the spec file and enjoy that all tests are green!

```
$ pipenv run mamba tennis_spec.py

.

1 examples ran in 0.0022 seconds
```

1.2 Example Groups

1.2.1 Basic structure (description / it)

Mamba is an internal Python DSL for creating executable examples.

Mamba organizes execution around example groups, and tries to use a human like languge for expressing behaviour. Like a human conversation.

```
from mamba import description, context, it

with description('A topic') as self:
  with context('when more context is given'):
    with it('happens some stuff and I can check it'):
      value = 'hello'
      other = 'world'

      assert value != other
```

Mamba uses context managers for organizing the code, so the let's see the about these context managers:

- description: Creates a new example group.
- context: Is basically an alias to description, and also creates an example group. But its usage is restricted to nested example groups.
- it: Creates a new example. Code inside it, will be executed as a test.

Internally, mamba parses this code using the Python ast module, and generates a class for every example group and a method for every example. And executes these classes like unittest (by example!).

1.2.2 Helper methods

For supporting a nice experience writing examples, mamba allows defining helper methods.

```
from mamba import description, context, it

with description('Refactoring Goodies') as self:
    with it('allows calling a defined method inside the example group'):
      assert self.hello('python') != self.hello('pandas')

    def hello(self, world):
      return 'hello, %s'.format(world)
```

You can define a helper method using Python language structures. And these helpers are exposed to example in the same group and in nested groups within that group, but not on parents or sibling groups.

1.2.3 Pending examples

Sometimes I've found myself keeping a small ToDo record about what should test next. Or perhaps we need to disable an specific test.

Mamba supports these cases using pending examples:

```
from mamba import description, _it

with description('Pending Examples') as self:
    with _it('will not run any pending example (marked with an underscore)'):
      assert False
```

When running this spec, we get the following output:

```
*

0 examples ran (1 pending) in 0.0003 seconds
```

And this also works with example groups:

```
from mamba import description, _context, it

with description('Pending Examples') as self:
    with _context('when running a pending context (marked with an underscore)'):
      with it('will not run any example under a pending context'):
        assert False

      with it('will not be run either'):
        assert False
```

And when executing this spec:

```
**

0 examples ran (2 pending) in 0.0005 seconds
```

1.2.4 Shared contexts

In order to DRY up your specs it is possible to define shared contexts.

```
from mamba import shared_context, it

with shared_context('Shared store examples'):
    with it('can retrieve stored items'):
        item = {'name': 'Bob'}
        self.store.add(id=1, item=item)

        assert self.store.get(id=1) == item
```

The examples in a shared context are not executed. You need to include the context by using the *included_context* context manager with the exact same description as the defined shared context.

```
from mamba import shared_context, included_context, it, before, describe

from app.store import InMemoryStore, SQLStore

with shared_context('Shared store examples'):
    with it('can retrieve stored items'):
        item = {'name': 'Bob'}
        self.store.add(id=1, item=item)
```

(continues on next page)

(continued from previous page)

```
        assert self.store.get(id=1) == item

with describe(InMemoryStore):
    with before.each:
        self.store = InMemoryStore()

    with included_context('Shared store examples'):
        pass

with describe(SQLStore):
    with before.each:
        self.store = SQLStore(host='localhost')

     with included_context('Shared store examples'):
        pass
```

Any examples and example groups defined inside the *included_context* block are **added** to the ones of the previously defined context. Any hooks defined inside the *included_context* **overwrite** those defined in the *shared_context*.

1.3 Hooks

1.3.1 'before' and 'after' hooks

Sometimes several tests shares some code. For avoiding repeating same code over and over, we could use *before* and *after* context manager.

These hooks are executed before or after every example or every example group.

```
from mamba import description, before, after, it

with description('Hooks') as self:

  with before.all:
    # This code is executed once, before executing any examples in this group

  with before.each:
    # This code is executed before every example

  with after.all:
    # This code is executed after all of the examples in this group

  with after.each:
    # This code is executed after each example
```

A more realistic example would be:

```
from mamba import description, before, it

class Stuff(object):

  def __init__(self):
    self._elements = []

  def elements(self):
```

(continues on next page)

(continued from previous page)

```
    return self._elements

  def add_element(self, element)
    self._elements.append(element)

with description(Stuff) as self:

  with before.each:
    # Initialize a new stuff for every example
    self.stuff = Stuff()

  with it('has 0 elements'):
    expect(self.stuff.elements()).to(have_length(0))

  with it('accepts elements'):
    self.stuff.add_element(object())

    expect(self.stuff.elements()).to(have_length(1))
```

1.4 Filtering

Sometimes, constraining which examples are run is really useful. Try to think in testing pyramid:

- Unit Tests
- Integration Tests
- End2End Tests

E2E tests are slowest, but integration ones are slower than unit. Perhaps I would like to keep my unit tests running all time, for making the feedback loop shorter, and execute integration tests only a few times before committing.

Mamba supports this use case using tags.

1.4.1 Including examples with a filter

An inclusion filter is a filter which runs all tests which matches with a tag, let's see an example:

```
# customer_repository_spec.py

from mamba import description, before, it
from expects import expect, be_equal

from app import repositories, infrastructure, model

with description(repositories.CustomerRepository, 'integration') as self:
  with before.each:
    connection = intrastructure.create_connection()
    self.database_cleaner = infrastructure.DatabaseCleaner(connection)
    self.database_cleaner.clean()

    self.repository = repositories.CustomerRepository(connection)
```

(continues on next page)

(continued from previous page)

```
  with it('stores a new customer'):
    customer = model.Customer()

    self.repository.put(customer)
    retrieved = self.repository.find_by_id(customer.id)

    expect(customer).to(be_equal(retrieved))
```

This is a contract test. It uses database and cleans all records on every execution. Potentially is a bit more expensive than other example which uses a fake-implementation in memory.

```
# customer_spec.py

from mamba import description, before, it
from expects import expect, be_equal

from app import model

with description(model.Customer, 'unit') as self:
  with before.each:
    self.customer = Customer()

  with it('adds orders'):
    customer.add_order(model.Order('Implementing Domain-Driven Design'))

    expect(customer).to(be_equal(retrieved))
```

So you are able to run all tests:

```
$ pipenv run mamba
```

Or run only unit tests:

```
$ pipenv run mamba -t unit
```

Or tun only integration tests

```
$ pipenv run mamba -t integration
```

You can select the inclusion tag using the -t parameter in command line.

1.4.2 Focused examples

This is an special case of example inclusion. This allows to focus execution only in an example or an example group. Sometimes you will need to focus execution only in a small piece of code for a while.

```
from mamba import description, it, fit

from katas import MarsRover

with description(MarsRover) as self:
  with context('when starts at 0, 0 and facing north'):
    with before.each:
      self.mars_rover = MarsRover((0, 0), 'N')
```

(continues on next page)

(continued from previous page)

```
    with it('moves north'):
      self.mars_rover.move('N')

      expect(self.mars_rover.position()).to(be_equal((0, 1))
      expect(self.mars_rover.direction()).to(be_equal('N'))

    # This is the unique example that will be executed
    with fit('moves east'):
      self.mars_rover.move('E')

      expect(self.mars_rover.position()).to(be_equal((1, 0))
      expect(self.mars_rover.direction()).to(be_equal('E'))
```

So when running this example:

```
$ pipenv run mamba

.

1 examples ran in 0.0014 seconds
```

It only runs the 'moves east' example. And this could be applied to example groups too, using the *fcontext* context manager.

And please, be kind with your teammates and avoid committing focused example. Eventually they will blame me for this ;)

1.5 Formatters

Mamba bundles with two formatters.

1.5.1 Progress Formatter

This is the default formatter. It displays a point for every example executed:

- Green point: It means that example has passed
- Yellow point: It means that example has been skipped
- Red point: It means that example has failed

1.5.2 Documentation Formatter

This is an extra formatter that allows you to read in a tree way. It uses same color scheme than previous formatter.

Is useful when you have a few tests and want to check that examples are written in human language.

For enabling it:

```
$ pipenv run mamba --format=documentation
```

1.5.3 Custom Formatters

Mamba supports third party formatters. Imagine there is a new IDE or some specific needs for your Continuous Integration tool, so:

```
$ pipenv run mamba --format=wondertech.MyCustomFormatter
```

And mamba tries to instantiate the *wondertech.MyCustomFormatter* class. But there are 2 conditions that should be met:

- A settings object is passed to object constuctor
- Inherit from mamba.formatter.Formatter for overriding methods

1.6 Integration with Other Libraries

1.6.1 Assertion Libraries

Mamba is "just" a DSL and a test runner, it does not include any assertion mechanism. It should work with any assertion library.

Although, my **personal preference** is using it with expects:

```
from mamba import description, it
from expects import expect, be_equal

with description('Assertion libraries'):
  with it('can be used with expects'):
    expect(True).to(be_true)
```

Or do you prefer to use Hamcrest assertions?

```
from mamba import description, it
from hamcrest import assert_that, is_

with description('Assertion libraries'):
  with it('can be used with hamcrest'):
    assert_that(True, is_(True))
```

So, you should be able to use mamba with your preferred assertion library: should_dsl, sure or even with plain Python assertions.

1.6.2 Test Doubles Libraries

Same that last point here. Mamba does not preescribe any test double library, it should work with any library.

Another time, my **personal preference** is using Doublex and doublex-expects:

```
from mamba import description, it
from expects import expect
from doublex import Spy
from doublex-expects import have_been_called

with description('Test Doubles Libraries'):
  with it('can be used with doublex'):
```

(continues on next page)

(continued from previous page)

```
    with Spy() as sender:
      sender.is_usable_with_doublex().returns(True)

    mail_gateway = Mailer(sender)
    mail_gateway.send('Hello')

    expect(sender.is_usable_with_doublex).to(have_been_called)
```

You prefer mockito?

```
from mamba import description, it
from expects import expect, be_true
from mockito import mock

with description('Test Doubles Libraries'):
  with it('can be used with mockito'):
    sender = mock()

    when(sender).is_usable_with_mockito().returns(True)

    expect(sender.is_usable_with_mockito()).to(be_true)
```

So, you can use mamba with your peferred test doubles library too: python mock, or even hand crafted fake objects.

1.7 Other Features

There are other features that does not fit in other categories. Just a little explanation.

1.7.1 Slow examples

Mamba is able to highlight an example when is tooking more time than a threeshold. By default is 0.75 seconds, but you can control with the command line:

```
$ pipenv run mamba --slow 0.5
```

This will highlight all tests that takes more than 0.5 seconds.

1.7.2 Coverage

Mamba can be used with coverage tool.

```
$ pipenv run mamba --enable-coverage
```

And this generate a .coverage file and you can generate a HTML report:

```
$ pipenv run coverage html
```