MiniBench Documentation

Release 0.1.2

Axel Haustant

November 21, 2015

Contents

MiniBench provides a simple framework for benchmarking following the `unittest` module pattern.

You can install minibench with pip:

```
$ pip install minibench
```

Then, you just have to write a `.bench.py` file.

```
# fake.bench.py
from minibench import Benchmark

class FakeBenchmark(Benchmark):
    '''Fake benchmark'''
    def bench_fake(self):
        '''Run my bench'''
        # Do something
```

Then run it with the `bench` command

```
$ bench
>>> Fake benchmark (x5)
Run my bench ..........................................  (0.1234s)
```

Contents:

CHAPTER 1

Quickstart

You can install minibench with pip:

```
$ pip install minibench
```

Write your benchmarks as you would write you unittests. Just create `.bench.py` file.

```
# fake.bench.py
from minibench import Benchmark

class FakeBenchmark(Benchmark):
    '''Fake benchmark'''
    def bench_fake(self):
        '''Run my bench'''
        # Do something

    def bench_another(self):
        '''Another bench'''
        # Do something
```

Then run it with the `bench` command

```
$ bench
>>> Fake benchmark (x5)
Run my bench ..............................  (0.1234s / 0.1233s)
Another bench .............................  (0.1234s / 0.1233s)
```

CHAPTER 2

Writing benchmarks

2.1 Basics

Writing a benchmark is as simple as extenting *Benchmark*. Each method will be run *times* times. A benchmark method should start with `bench_`.

```
from minibench import Benchmark


class SumBenchmark(Benchmark):
    times = 1000

    def bench_sum(self):
        sum(x for x in range(5))

    def bench_consecutive_add(self):
        total = 0
        for x in range(5):
            total += x
```

```
$ bench examples/sum.bench.py
Running 1 benchmark
--------------------
>>> Sum benchmark (x1000)
Consecutive add...................  0.00142s / 0.00000s
Sum...............................  0.00245s / 0.00000s
 Done
```

2.2 Documenting

Documenting you benchmark is as simple as writing explicit docstrings. Only the first line will be kept.

```
from minibench import Benchmark


class SumBenchmark(Benchmark):
    '''
    A simple sum benchmark

    This will be ignored.
```

```
    '''
    times = 1000

    def bench_sum(self):
        '''Sum using sum()'''
        sum(x for x in range(5))

    def bench_consecutive_add(self):
        '''Sum using +='''
        total = 0
        for x in range(5):
            total += x
```

```
$ bench examples/sum.bench.py
Running 1 benchmark
-------------------
>>> A simple sum benchmark (x1000)
Sum using sum()............................  0.00142s / 0.00000s
Sum using +=...............................  0.00245s / 0.00000s
 Done
```

2.3 Hooks

The *Benchmark* provide some hooks as methods:

- *before_class()*: executed once before each class
- *before()*: executed once before each method
- *before_each()*: executed before each method call
- *after_class()*: executed once after each class
- *after()*: executed once after each method
- *after_each()*: executed after each method call

```
from minibench import Benchmark

class MyBench(Benchmark):

    def before_class(self):
        # Will be executed once before all class methods
        pass

    def before(self):
        # Will be executed once before each method
        pass

    def before_each(self):
        # Will be executed before each method call
        pass

    def after_class(self):
        # Will be executed once after all class methods
        pass

    def after(self):
```

```
        # Will be executed once after each method
        pass

    def after_each(self):
        # Will be executed aftereach method call
        pass
```

CHAPTER 3

Console Client

MiniBench comes with a console client *bench*.

This client takes a list of glob patterns as parameters. If none is provided, it default on `**/*.bench.py`.

```
$ bench
$ bench benchmarks/*.bench.py
$ bench benchs/*.bench tests/*.bench
```

3.1 Override Times

You can overrides how many `times` a methods are called with the `--times` option

```
$ bench -t 10000
$ bench --times 1000
```

3.2 Export reports

You can export the result summary to the following formats:

- JSON
- CSV
- reStructuredText
- Markdown

```
$ bench --json out.json --csv out.csv
$ bench --rst out.rst --md out.md
```

3.3 Run against a reference

MiniBench provides an easy to compare results against a previous JSON report with the `--ref` option.

```
$ bench --json out.json -t 100
Running 1 benchmark
--------------------
>>> Glob benchmark (x100)
```

```
Fnmatch.....................................  1.61257s / 0.01613s
Glob........................................  2.02383s / 0.02024s
Re..........................................  1.39118s / 0.01391s
 Done

$ bench --ref out.json -t 100
Running 1 benchmark
--------------------
>>> Glob benchmark (x100)
Fnmatch.............................  1.60748s / 0.01607s (-0.31%)
Glob................................  1.97594s / 0.01976s (-2.36%)
Re..................................  1.48161s / 0.01482s (+6.50%)
 Done

$ bench --ref out.json -t 100 --unit seconds
Running 1 benchmark
--------------------
>>> Glob benchmark (x100)
Fnmatch................  1.60748s / 0.01607s (-0.00508s / -0.00005s)
Glob...................  1.97594s / 0.01976s (-0.04788s / -0.00048s)
Re.....................  1.48161s / 0.01482s (+0.09043s / +0.00090s)
 Done
```

3.4 Debug mode

By default MiniBench does not stop on error nor display it. If you want to stop on first error and display it, you shoud run in debug mode with the `-d` or the `--debug` option.

```
$ bench -d
$ bench --debug
```

In debug mode, it will not continue to execute a failing method and it will display the raised error.

3.5 Bash completion

A bash completion script is provided in the minibench github repository: bench-complete.sh.

CHAPTER 4

Reporters

Reporters provide a simple way of formatting your benchmark reports. They all inherit from *BaseReporter*.

CHAPTER 5

API

This part of the documentation lists the full API reference of all public classes and functions.

5.1 Core

class minibench.**Benchmark**(*times=None*, *prefix=u'bench_'*, *debug=False*, *before=None*, *before_each=None*, *after=None*, *after_each=None*, ***kwargs*)

Base class for all benchmark suites

times = The number of iteration to run each method

after()

Hook called once after each method

after_class()

Hook called after each class

after_each()

Hook called after each method once

before()

Hook called once before each method

before_class()

Hook called before each class

before_each()

Hook called before each method

label

A human readable label

label_for(*name*)

Get a human readable label for a method given its name

run()

Collect all tests to run and run them.

Each method will be run *Benchmark.times*.

class minibench.**RunResult**(*duration*, *success*, *result*)

Store a single method execution result

class minibench.**BenchmarkRunner**(**filenames*, ***kwargs*)

Collect all benchmarks and run them

Parameters

- **filenames** (*string*) – the benchmark files names
- **reporters** (*list*) – the reporters classes or instance to run
- **debug** (*bool*) – Run in debug mode if `True`

load_from_module(*module*)
Load all benchmarks from a given module

load_module(*filename*)
Load a benchmark module from file

run(***kwargs*)
Run all benchmarks.

Extras kwargs are passed to benchmarks construtors.

5.2 Reporters

class minibench.**BaseReporter**(*precision=5*, ***kwargs*)
Base class for all reporters

after_class(*bench*)
Hook called once after each benchmark class

after_method(*bench*, *method*)
Hook called once after each benchmark method

before_class(*bench*)
Hook called once before each benchmark class

before_method(*bench*, *method*)
Hook called once before each benchmark method

end()
Hook called once on run end

key(*bench*)
Generate a report key from a benchmark instance

progress(*bench*, *method*, *times*)
Hook called after each benchmark method call

start()
Hook called once on run start

summary()
Compute the execution summary

class minibench.**FileReporter**(*filename*, ***kwargs*)
A reporter dumping results into a file

Parameters **filename** (*string*) – the output file name

end()
Dump the report into the output file.

If the file directory does not exists, it will be created. The open file is then given as parameter to `output()`.

line(*text=u''*)
A simple helper to write line with ' '

output(*out*)
Serialize the report into the open file.

Child classes should implement this method.

Parameters **out** (*file*) – an open file object to serialize into.

class minibench.**JsonReporter**(*filename*, ***kwargs*)
A reporter dumping results into a JSON file

Parameters **filename** (*string*) – the output file name

class minibench.**CsvReporter**(*filename*, ***kwargs*)
A reporter dumping results into a CSV file

The CSV will have the following format:

Benchmark	Method	Times	Total (s)	Average (s)

It uses *;* character as delimiter and *"* as delimiter. All Strings are quoted.

Parameters **filename** (*string*) – the output file name

class minibench.**MarkdownReporter**(*filename*, ***kwargs*)
A reporter rendering result as a markdown table.

Each benchmark will be rendered as a table with the following format:

Method	Times	Total (s)	Average (s)

Parameters **filename** (*string*) – the output file name

class minibench.**RstReporter**(*filename*, ***kwargs*)
A reporter rendering result as a reStructuredText table

Each benchmark will be rendered as a table with the following format:

Method	Times	Total (s)	Average (s)

Parameters **filename** (*string*) – the output file name

class minibench.cli.**CliReporter**(*ref=None*, *debug=False*, *unit='%'*, ***kwargs*)
A reporter that display running benchmarks with ANSI colors

CHAPTER 6

Changelog

6.1 0.1.2 (2015-11-21)

- Improve diff from reference and added `-u/--unit` option
- Properly handle precision and added `-p/--precision` option

6.2 0.1.1

- Fix Python 3 Wheel packaging

6.3 0.1.0

- Initial release

CHAPTER 7

Contributing

MiniBench is open-source and very open to contributions.

7.1 Submitting issues

Issues are contributions in a way so don't hesitate to submit reports on the official bugtracker.

Provide as much informations as possible to specify the issues:

- the minibench version
- a stacktrace
- installed applications list
- a code sample to reproduce the issue
- ...

7.2 Submitting patches (bugfix, features, ...)

If you want to contribute some code:

1. fork the official minibench repository
2. create a branch with an explicit name (like `my-new-feature` or `issue-XX`)
3. do your work in it
4. rebase it on the master branch from the official repository (cleanup your history by performing an interactive rebase)
5. submit your pull-request

There are some rules to follow:

- your contribution should be documented (if needed)
- your contribution should be tested and the test suite should pass successfully
- your code should be mostly PEP8 compatible with a 120 characters line length
- your contribution should support both Python 2 and 3 (use `tox` to test)

You need to install some dependencies to develop on minibench:

```
$ pip install -e .[test,dev]
```

An Invoke `tasks.py` is provided to simplify the common tasks:

```
$ inv -l
Available tasks:

  all          Run tests, reports and packaging
  clean        Cleanup all build artifacts
  completion   Generate bash completion script
  cover        Run tests suite with coverage
  dist         Package for distribution
  doc          Build the documentation
  qa           Run a quality report
  test         Run tests suite
  tox          Run test in all Python versions
```

To ensure everything is fine before submission, use `tox`. It will run the test suite on all the supported Python version and ensure the documentation is generating.

```
$ tox
```

You also need to ensure your code is compliant with the minibench coding standards:

```
$ inv qa
```

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

m