Python-Project-Skeleton

Release 0.11.3

Joao MC Teixeira

Feb 11, 2023

CONTENTS

CHAPTER

ONE

PYTHON PACKAGE SKELETON TEMPLATE

1.1 Summary

This repository is a **skeleton template** for a **Python application/library** compliant with the latest team-development and software deployment practices within a continuous integration (CI) framework. You can use this repository as a source of information and a resource to study CI. Also, you can use this repository as a direct template for your repositories.

Note that this repository reflects the setup I like the most and that covers the CI needs for my Python projects, which include:

- **A robust Python library/application file hierarchy with packages, modules, clients, and documentation:**
  - detailed, yet simple, `setup.py`
  - retrievable `README` and `CHANGELOG`
  - documentation deployed in ReadTheDocs
  - the unusual adoption of the `src` directory layer (love it)
  - examples of packages and modules hierarchy
  - examples of Python command-line interfaces
- **A unique testing framework for developers with tox and pytest**
  - guarantees tests are reproducible for all developers
  - ensures same lint rules are always applied (local and remotely)
  - ensures all desired Python versions are covered
  - adopts hypothesis
- **Fully automated continuous integration services with GitHub Actions**
  - automatic testing on Linux, MacOS, and Windows
  - automatic testing simulated upon deployment with `tox`

- test coverage report to Codecov
- automated version bump with bump2version, git tagging, and Python packaging to PyPI on Pull Request merge

1.2 Motivation

I developed this repository to understand how to implement the best practices for team-based development and automatic deployment of a scientific software written in Python. Nonetheless, I believe the strategy reviewed here can be applied to most general-purpose Python libraries.

This repository does **not** intend to be a cookiecutter-like repository. There are very well documented cookiecutters, even for scientific software, if you are looking for one of those. However, when I started developing Python libraries, I decided that blindly using a cookiecutter would not provide me the learning insights from configuring CI services because I would miss the details of what was actually being implemented. Hence, assembling this *template* from scratch as a full working repository was my best approach to obtain a useful understanding of CI. Now, this repository serves as a reference guide for all my projects and hopefully will serve you, too. I keep constantly updating this repository, expect one to two updates/reviews per year.

1.3 Acknowledgments

I want to acknowledge ionel discussions about *Packaging a python library*. They are a pillar in my understanding and decisions on this matter, and I really recommend reading his blog post and references herein.

I configured the CI pipeline to my needs by taking bits and pieces from many places. Kudos to python-nameless and cookiecutter-pylibrary; two primary sources of information for the *python-project-skeleton* repository, especially in the first versions using Travis and Appveyor.

When migrating to GitHub Actions, I based my choices on the version bump and deploying workflows @JoaoRodrigues assembled for pdb-tools; on the tox-gh-actions package; and on structlog. Implementation details have evolved with newest versions.

I refer to other important sources of information as comments in the specific files. Thanks, everyone, for keeping open discussions on internet.

1.4 How to use this repository

The documentation pages explain how to use this template for your projects and the implementation details adopted here. The documentation pages also serve to demonstrate how to compile documentation with Sphinx and deploy it online with ReadTheDocs.

1.5 Issues and Discussions

As usual for any GitHub-based project, raise an issue if you find any bug or want to suggest an improvement, or open a discussion if you want to discuss or chat :wink:

1.6 Projects using this skeleton

Below, a list of the projects using this repository as template or as base for their CI implementations:

- julie-forman-kay-lab/IDPConformerGenerator
- haddocking/HADDOCK3
- THGLab/MCSCE
- joaomcteixeira/taurenmd
- MDAnalysis/mdacli

If you use this repository as a reference for your works, let me know, so I list your work above, as well.

1.7 Version

v0.11.3

CHAPTER

TWO

THE RATIONALE BEHIND THE PROJECT

In the following sections I explain the rationale I used to configure this template repository. Also, these are the same strategies I adopt in my current projects. In summary, the following sections represent my view on software development practices, either it is open or closed source, done in teams or by single individuals. Yes, I also follow these *rules* when I am the sole developer.

2.1 Branch organization

This repository complies with the ideas of agile development. Why? Because I found the agile strategy to fit best the kind of projects I develop. Is this the best strategy for your project? Well, you have to evaluate that.

In the "python-project-skeleton" layout, there is only one production branch, the *main* branch, representing the latest state of the software. This layout contrasts with the opposite strategy where the *stable* and the *latest* branches are kept separate. For the kind of projects I develop, keeping the *stable* and the *latest* separated offers no benefit. Honestly, I believe such separation defeats the purpose of agile development with Semantic Versioning 2, which is the versioning scheme I adopted here, and in most of my projects.

2.2 Versioning

The versioning scheme adopted here is the Semantic Versioning 2.

2.3 The development process

Any code addition must come in the form of pull requests from forked repositories. Any merged pull request is followed the by respective increment in the software version and consecutive deployment of the new version in PyPI. If there are 5 pull request merges in one day, there will be 5 new versions releases that same day. Each new version should be tagged accordingly in git.

Additions to the documentation *also* count to the version number.

Additions to the CI platform *may* skip an increase in version number, using the `[SKIP]` tag in the merge commit message, as explained in *version release*.

CHAPTER

THREE

HOW TO USE THIS TEMPLATE

You can use the `python-project-skeleton` as a template for your own repositories thanks to the template feature of GitHub, *(Yes, you could use a cookiecutter, but this is not a coockiecutter).*

3.1 Changing names

Once you have created a new repository from this template, these are the names you need to change in the files to adapt the template to you and your project:

Note: If you are using bash, you can use: `grep -rliI "text to search" .` to identify which files have the string you want to change.

Note: In bash, use the following to replace a string in multiple files. `find ./ -type f -exec sed -i -e 's/apple/orange/g' {} ;` Kudos to https://stackoverflow.com/questions/6758963.

1. Where it says `joaomcteixeira`, rename it to your GitHub user name. In `setup.py` and in `AUTHORS.rst` there is also my full written name, rename it to yours.

2. Where ever it says `sampleproject` change it for the name of your project. That is, what you would like your users to import.

```
import sampleproject
```

3. In `setup.py` rename `jmct-sampleproject` for the name your project will have at PyPI. Usually this is the same as the name you decided above.

4. Replace `python-project-skeleton` to the name of the repository of your project. It might be the same name as the Python package and the PyPI package, that is up to you entirely.

5. This template is deployed at `test.pypi.org`. You need to change the last line of the version-bump-and-package.yml file and remove the `--repository testpypi` tag so that your project is deployed at the main PyPI repository.

6. Remove and edit some of the unnecessary files in the `docs` folder. Mainly those related explicitly to this template repository and reference herein.

Despite you can use the find/replace commands, I suggest you navigate every file and understand its functionality and which parameters you need to change to fit the template to your project. They are not that many files.

Please read the *Template Configuration* section for all the details on the different workflows implemented.

Finally, you can remove those files or lines that refer to integrations that you do not want. For example, if you don't want to have your project tracked by CodeClimate, remove the files and lines related to it.

Enjoy, and if you like this template, please let me know.

CHAPTER

FOUR

TEMPLATE CONFIGURATION

I will explain how the continuous integration is configured and, consequently, how you can configure those same services for you repository.

4.1 Python Project Layout

The file structure for this Python project follows the `src`, `tests`, `docs` and `devtools` folders layout.

4.1.1 The src layout

I discovered storing the project's source code underneath a `src` directory layer instead of directly in the project's root folder is one of the most controversial discussions regarding the organization of Python projects. Here, I adopted the `src`-based layout discussed by ionel in his blog post. After more than one year of using `src`, I have nothing to complain about; on the contrary, it saved many issues related to `import` statements. Either importing from the repository or the installed version? The `src` guarantees stable imports. More on ionel's blog (see also src-nosrc example).

The `src/sampleproject` folder hosts the actual source of the project. In the current version of this template, I don't discuss how to organize a source code of a project. I am looking forward doing that in future versions.

4.1.2 Testing

Here, tests are encapsulated in a separate `tests` folder. With this encapsulation, outside the main library folder, it is easier to control that tests do not import from relative paths and can only access the library code after library installation (regardless of the installation mode). Also, having `tests` in a separated folder facilitates the configuration files layout on excluding tests from deployment (`MANIFEST.in`) and code quality (`.codacy.yaml`) or coverage (`.coveragerc`).

4.1.3 Documentation

All documentation related files are stored in a `docs` folder. Files in `docs` will be compiled using Sphinx to generate the HTML documentation web pages. You can follow the file and folder structure in this repository as an example of how to assemble the documentation. You will see files that contain text and others that import text from relative files. The latter strategy avoids repeating information.

4.1.4 devtools

The `devtools` folder hosts the files related to development. In this case, I used the idea explained by Chodera Lab in their structuring your project guidelines.

4.2 Continuous Integration

Here, I overview the implementation strategies for the different continuous integration and quality report platforms. In the previous versions of this skeleton template, I used Travis-CI and AppVeyor-CI to run the testing workflows. In the current version, I have migrated all these operations to GitHub Actions.

Does this mean you should not use Travis or AppVeyor? *Of course not.* Simply, the needs of my projects and the time I have available to dedicate to CI configuration do not require a dedicated segregation of strategies into multiple servers and services and I can perfectly accommodate my needs in GitHub actions.

Are GitHub actions better than Travis or AppVeyor? I am not qualified to answer that question.

When using this repository, keep in mind that you don't need to use all services adopted here. You can drop or add any other at your will by removing the files or lines related to them or adding new ones following the patterns presented here.

The following list summarizes the platforms adopted:

1. **Building and testing**
    - GitHub actions
2. **Quality Control**
    - Codacy
    - Code Climate
3. **Test Coverage**
    - Codecov
4. **Documentation**
    - Read the Docs

I acknowledge the existence of many other platforms for the same purposes of those listed. I chose these because they fit the size and scope of my projects and are also the most used within my field of work.

4.2.1 Configuring GitHub Actions

You may wish to read about GitHub actions. Here, I have one Action workflow per environment defined in `tox`. Each of these workflows runs on each of the python supported versions and OSes. These tests regard unit tests, documentation build, lint, integrity checks, and, finally, version bump and package deploying on PyPI.

The CI workflows trigger every time a new pull request is created and at each commit to that PR. However, the `lint` tests do not trigger when the PR is merged to the `main` branch. On the other hand, the `version bump` workflow triggers only when the PR is accepted.

In this repository you can find two PRs demonstrating: one where tests pass and another where tests fail.

Version release

Every time a Pull Request is merged to *main* branch, the deployment workflow triggers. This action bumps the new version number according to the merge commit message, creates a new GitHub tag for that commit, and publishes in PyPI the new software version.

As discussed in another section, here I follow the rules of Semantic Versioning 2.

If the Pull Request merge commit starts with `[MAJOR]`, a major version increase takes place (attention to the rules of SV2!), if a PR merge commit message starts with `[FEATURE]` it triggers a *minor* update. Finally, if the commit message as not special tag, a *patch* update is triggered. Whether to trigger a *major*, *minor*, or *patch* update concern mainly the main repository maintainer.

This whole workflow can be deactivate if the commit to the `main` branch starts with `[SKIP]`.

In conclusion, every commit to `main` without the `[SKIP]` tag will be followed by a version upgrade, new tag, new commit to `main` and consequent release to PyPI. You have a visual representation of the commit workflow in the Network plot.

How version numbers are managed?

There are two main version string handlers out there: bump2version and versioneer. I chose *bump2version* for this repository template. Why? I have no argument against *versioneer* or others, simply I found `bumpversion` to be so simple, effective, and configurable that I could only adopt it. Congratulations to both projects nonetheless.

4.2.2 Code coverage

`Codecov` is used very frequently to report test coverage rates. Activate your repository under `https://about.codecov.io/`, and follow their instructions.

Coverage reports are sent to Codecov servers when the `test.yml` workflow takes place. This happens for each PR and each merge commit to `maint`.

The .coveragerc file, mirrored below, configures `Coverage` reports.

```
[paths]
source =
    src
    */site-packages

[run]
branch = true
source =
    sampleproject
parallel = true

[report]
show_missing = true
precision = 2
omit = *migrations*
exclude_lines =
    if __name__ == .__main__.:
```

The `.coveragerc` can be expanded to further restraint coverage analysis, for example adding these lines to the `exclude` tag:

```
[report]
exclude_lines =
if self.debug:
pragma: no cover
raise NotImplementedError
if __name__ == .__main__.:
```

Remember that if you don't want to use these services, you can simply remove the respective files from your project.

4.2.3 Code Quality

Here, we have both `Codacy` and `Code Climate` as code quality inspectors. There are also others out there, feel free to suggested different ones in the Discussion tab.

Codacy

There is not much to configure for *Codacy* as far as this template is concerned. The only setup provided is to exclude the analysis of test scripts, this configuration is provided by the `.codacy.yaml` file at the root director of the repository. If you wish Codacy to perform quality analysis on your test scripts just remove the file or comment the line. Here we mirror the .codacy.yaml file:

```
---
exclude_paths:
 - 'tests/**'
```

Code Climate

There is not much to configure for Code Climate, as well. The only setup provided here is to exclude the analysis of test scripts and other *dev* files Code Climate checks by default, the `.codeclimate.yml` file at the root directory of the repository configures this behavior (look at the bottom lines). If you wish Code Climate to perform quality analysis on your test scripts just remove the file or comment the line.

Code Climate provides a **technical debt** percentage that can be retrieved nicely with *Badges*.

4.3 Read the Docs

Activate your project at Read the Docs platform (RtD), their web interface is easy enough to follow without further explanations. If your documentation is building under the *tox workflow* it will build in at Read the Docs.

4.3.1 Docs Requirements

Requirements to build the documentation page are listed in `devtools/docs_requirements.txt`:

```
sphinx>=2.2
sphinx-py3doc-enhanced-theme
sphinx-argparse
CommonMark
mock
```

Here we use Sphinx as the documentation builder and the sphinx-py3doc-enhanced-theme as theme, though you can use many different theme flavors, see Sphinx Themes.

4.3.2 Build version

By default, RtD has two main documentation versions (also called builds): the *latest* and the *stable*. The *latest* points to the `master` branch while the *stable* points to the latest GitHub tag. However, as we have discussed in *The rationale behind the project* section, here we keep only the *latest* version (that of the `master` branch) and other versions for the different releases of interest.

4.3.3 Google Analytics

Read the Docs allows straight forward implementation of Google Analytics tracking in the project documentation, just follow their instructions.

4.3.4 Local Build

The `[testenv:docs]` in `tox.ini` file simulates the RtD execution. If that test passes, RtD should pass.

To build a local version of the documentation, go to the main repository folder and run:

```
tox -e docs
```

The documentation is at `dist/docs/index.html`. The `tox` run also reports on inconsistencies and errors. If there are inconsistencies or errors in the documentation build, the PR won't pass the CI tests.

4.4 Badges

Badges point to the current status of the different Continuous Integration tools in the `master` branch. You can also configure badges to report on other branches, though. Are tests passing? Is the package building properly? Is documentation building properly? What is the quality of the code? Red lights mean something is wrong and you should attend it shortly!

Each platform provide their own badges, and you can modify and emulate the badge strategy in the `README.rst` file. Yet, you can further tune the badges style by creating your own personalized badge with Shields.io.

You have noticed already that there is no badge for PyPI. That is because "python-project-skeleton" is deployed at `test.pypi.org`. You will find also at Shields.io how to add the PyPI badge.

4.5 Configuration Files

4.5.1 MANIFEST.in

The `MANIFEST.in` file configures which files in the repository/folder are grabbed or ignored when assembly the distributed package. You can see that I package the `src`, the `docs`, other `*.rst` files, the `LICENSE` and nothing more. All configuration files, tests, and other Python related or IDE related files are excluded.

There is a debate on whether tests should be deployed along with the library source. Should they? Tox and the CI integration guarantees tests are run on *installed* versions of the software. So, I am the kind of person that considers there is no need to deploy tests alongside with the source. Users aren't going to run tests. Developers will.

Let me know if you think differently.

It is actually easy to work with MANIFEST.in file. Feel free to add or remove files as your project needs.

4.5.2 tox.ini

Tox configuration file might be the trickiest one to learn and operate with until you are familiar with tox's workflows. Read all about tox in their documentation pages. The `tox.ini` file contains several comments explaining the implementations I adopted.

4.5.3 bumpversion.cfg

The `.bumpversion.cfg` configuration is heavily related with the GitHub Actions workflows for automated packaging and deployment and the `CONTRIBUTING.rst` instructions. Specially, the use of commit and tag messages, and the trick with CHANGELOG.rst.

I have also used bumpversion in other projects to update the version on some badges.

CHAPTER

FIVE

INSTALLATION

This is an example page for a real project. In this page you can describe the installation steps required for end-users, use the *Contribution page* to provide the guidelines for developers.

5.1 Installation Example

At the command line:

```
pip install sampleproject
```

CHAPTER

SIX

USAGE

This is an example page for a real project. Describe here examples on how to use your software!

To use `sampleproject`:

```
import sampleproject
```

CHAPTER

SEVEN

CONTRIBUTING

Note: Here, I explain how to contribute to a project that adopted this template. Actually, you can use this same scheme when contributing to this template. If you are completely new to `git` this might not be the best beginner tutorial, but will be very good still ;-)

You will notice that the text that appears is a mirror of the `CONTRIBUTING.rst` file. You can also point your community to that file (or the docs) to guide them in the steps required to interact with you project.

There are several strategies on how to contribute to a project on github. Here, I explain the one I use for all the project I am participating. You can use this same strategy to contribute to this template or to suggest contributions to your project.

7.1 Fork this repository

Fork this repository before contributing. It is a better practice, possibly even enforced, that only pull request from forks are accepted. In my opinion enforcing forks creates a cleaner representation of the contributions to the project.

7.1.1 Clone the main repository

Next, clone the main repository to your local machine:

```
git clone https://github.com/joaomcteixeira/python-project-skeleton.git
cd python-project-skeleton
```

Add your fork as an upstream repository:

```
git remote add myfork git://github.com/YOUR-USERNAME/python-project-skeleton.git
git fetch myfork
```

7.2 Install for developers

Create a dedicated Python environment where to develop the project.

If you are using `pip` follow the official instructions on Installing packages using pip and virtual environments, most likely what you want is:

```
python3 -m venv newenv
source newenv/bin/activate
```

If you are using Anaconda go for:

```
conda create --name newenv python=3.7
conda activate newenv
```

Where `newenv` is the name you wish to give to the environment dedicated to this project.

Either under *pip* or *conda*, install the package in `develop` mode. Install also *tox*.

```
python setup.py develop
pip install tox
```

This configuration, together with the use of the `src` folder layer, guarantees that you will always run the code after installation. Also, thanks to the `develop` flag, any changes in the code will be automatically reflected in the installed version.

7.3 Make a new branch

From the `main` branch create a new branch where to develop the new code.

```
git checkout main
git checkout -b new_branch
```

Note the `main` branch is from the main repository.

Develop the feature and keep regular pushes to your fork with comprehensible commit messages.

```
git status
git add (the files you want)
git commit (add a nice commit message)
git push myfork new_branch
```

While you are developing, you can execute `tox` as needed to run your unit tests or inspect lint, or other integration tests. See the last section of this page.

7.3.1 Update your branch

It is common that you need to keep your branch update to the latest version in the `main` branch. For that:

```
git checkout main  # return to the main branch
git pull  # retrieve the latest source from the main repository
git checkout new_branch  # return to your devel branch
git merge --no-ff main  # merge the new code to your branch
```

At this point you may need to solve merge conflicts if they exist. If you don't know how to do this, I suggest you start by reading the official docs

You can push to your fork now if you wish:

```
git push myfork new_branch
```

And, continue doing your developments are previously discussed.

7.3.2 Update CHANGELOG

Update the changelog file under `CHANGELOG.rst` with an explanatory bullet list of your contribution. Add that list right after the main title and before the last version subtitle:

```
Changelog
=========

* here goes my new additions
* explain them shortly and well

vX.X.X (1900-01-01)
-------------------
```

Also add your name to the authors list at `docs/AUTHORS.rst`.

7.3.3 Pull Request

Once you finished, you can create a pull request to the main repository, and engage with the community.

Before submitting a Pull Request, verify your development branch passes all tests as *described below* . If you are developing new code you should also implement new test cases.

7.4 Uniformed Tests with tox

Thanks to Tox we can have a unified testing platform that runs all tests in controlled environments and that is reproducible for all developers. In other words, it is a way to welcome (*force*) all developers to follow the same rules.

The `tox` testing setup is defined in a configuration file, the tox.ini, which contains all the operations that are performed during the test phase. Therefore, to run the unified test suite, developers just need to execute `tox`, provided tox is installed in the Python environment in use.

```
pip install tox
# or
conda install tox -c conda-forge
```

One of the greatest advantages of using `tox` together with the *src layout* is that unit test actually perform on the installed source (our package) inside an isolated deployment environment. In order words, tests are performed in an environment simulating a post-installation state instead of a pre-deploy/development environment. Under this setup, there is no need, in general cases, to distribute unit test scripts along with the actual source, in my honest opinion - see MANIFEST.in.

Before creating a Pull Request from your branch, certify that all the tests pass correctly by running:

```
tox
```

These are exactly the same tests that will be performed online in the Github Actions.

Also, you can run individual testing environments if you wish to test only specific functionalities, for example:

```
tox -e lint  # code style
tox -e build  # packaging
tox -e docs  # only builds the documentation
tox -e test  # runs unit tests
```

CHAPTER

EIGHT

SOURCE DOCUMENTATION

8.1 AModule

Main DOCSTRING for amodule.

With several lines.

amodule.**hello**()

Print 'hello module'.

8.2 Libs

General Libraries for the project.

8.2.1 Something

samplemodule that performs sample operations.

Contains:

- SampleClass

class sampleproject.libs.samplemodule.**SampleClass**

Documentation of the SampleClass.

classmethod false()

Docstrings should not start with Returns…

Nonetheless, returns False

static true()

Return True my friend.

sampleproject.libs.samplemodule.**this_is_and_undocumented_function**(*some_parameter*)

8.3 sampleproject

Initial documentation of SampleProject.

CHAPTER

NINE

CHANGELOG

9.1 v0.11.3 (2023-02-11)

- updated check *CHANGELOG.rst* and error message (#34)

9.2 v0.11.2 (2023-02-07)

- Corrected *major* version bump action (#32)

9.3 v0.11.1 (2022-07-14)

- Add a CI .yml for envs running only on PRs (#31)

9.4 v0.11.0 (2022-07-14)

- merge all tox envs in a single matrix (#30)

9.5 v0.10.1 (2022-07-05)

- Correct docs/config reference in `.readthedocs.yml`

9.6 v0.10.0 (2022-07-05)

- Update docs content and structure

9.7 v0.9.6 (2022-06-30)

- Update README

9.8 v0.9.5 (2022-06-29)

- Update CI YAML files:
- Add python 3.10
- reduce number of installs

9.9 v0.9.4 (2022-06-29)

- move pull request template to `.github` folder.

9.10 v0.9.3 (2022-06-29)

- Add code of conduct from Contributor Covenant v2.

9.11 v0.9.2 (2022-05-26)

- Update badges in *readme* and corresponding *master* rename to *main* in services.

9.12 v0.9.1 (2022-05-26)

- revert to *pr* testenv in *tox*
- Remove the empty line in changelog title

9.13 v0.9.0 (2022-05-26)

- update tox environments. Tests now run with *test* env.
- update/simplify github actions accordingly
- attempts correct changelog line conflict
- update requirements.txt now depending on *setup.py*
- drop python 3.6

9.14 v0.8.1 (2021-11-09)

- updates radon commands in *tox.ini*

9.15 v0.8.0 (2021-01-26)

- renames tox env *chlog* to a more general *prreqs*
- corrects details in *CONTRIBUTING.rst*

9.16 v0.7.0 (2021-01-24)

- increased granularity of tox environments
- new lint env
- new chlog env
- check renamed to build
- tox environment scopes separated into different actions
- new lint only dedicated action
- new pull request requirements action

9.17 v0.6.0 (2021-01-24)

- updates package check command, now uses twine
- add scripts to clean up dist files after `tox -e check`
- Adds checks to CHANGELOG
- Adds new tox env just for lint checks
- lint is kept in `check` env nonetheless
- updates documentation accordingly

9.18 v0.5.0 (2021-01-24)

- Adds documentation for some configuration files

9.19 v0.4.0 (2021-01-23)

- Updates documentation to the new Github Actions integration
- Updates documentation in general

9.20 v0.3.1 (2021-01-22)

- Synchronized CHANGELOG with `.bumpversion`

9.21 v0.3.0 (2021-01-22)

- simplifies `setup.py`
- defines rules for `CHANGELOG.rst`
- adds `check` `tox` env to py37 machine

9.22 v0.2.2 (2021-01-22)

- Updates CI framework to GitHub Actions
- adds action to automate version bump and package build to PyPI
- completes CI for Linux, Windows, and MacOS
- reports test coverage to Codecov
- updated/enhanced bump2version configuration
- bump2version also changes CHANGELOG

9.23 v0.2.1 (2020-05-31)

- updated tox to accepts posargs in *pytest* and *flake8*

9.24 v0.2.0 (2020-01-31)

- Implemented Travis-CI for Windows, MacOSX and Linux * for Python: 3.6, 3.7 and 3.8 * all previous without using anaconda expect for MacOSX 3.8 * I have nothing against Anaconda ;-), on the contrary, I use it everyday.
- Improved `tox.ini` workflow to my current favorite standards.
- Implemented mock strategy to avoid installing dependencies for documentation generation. * TOXENV docs

9.25 v0.1.0 (2019-10-03)

- First release on PyPI.

CHAPTER

TEN

AUTHORS

- Joao M. C. Teixeira (webpage, github)
- Kale Miller (kmgithub)

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

s

INDEX