Once Documentation

Release 1.0.0

Mika Pflüger

Sep 16, 2022

CONTENTS:

CHAPTER ONE

ONCE_ONLY

Run a python script or function only once in a given time frame.

If, for example, you have a script or service which might be called frequently, but you want to report errors only once daily to not annoy people too much, *once_only* is the library for you.

1.1 Quickstart

Suppose you want your script to complain, but not more than once a day. `once_only.daily` is the tool you need:

```
import once_only

@once_only.daily
def complain():
    print("This is not right!")
```

Now you can run `complain` as often as you want, from the same python script or from others, it will only be run at most once a day, so that at least 24 hours are between two complaints.

If you want to complain more or less often, there are other variants:

object	time delta
`once_only.weekly`	1 week
`once_only.daily`	1 day
`once_only.hourly`	1 hour
`once_only.minutely`	1 minute
`once_only.Once()`	custom `datetime.timedelta`

1.2 Advanced Usage

Instead of using a `once_only.Once` object as a decorator, you can also access it directly via the `check_ready()` and `check_ready_trigger()` functions:

```
import once_only
import datetime

once_every_two_hours = once_only.Once(datetime.timedelta(hours=2))

if once_every_two_hours.check_ready():
    print("More than two hours have passed since last run!")

if not_a_dry_run and once_every_two_hours.check_ready_trigger():
    print("Triggering timer and running!")
```

Note that all instances of `once_only.Once` with the same time delta share the same timer, but those with different time deltas don't share the timer. So, if you have never run anything before, this:

```
import once_only
import datetime

@once_only.minutely
def run_minutely():
    print("minutely")

@once_only.hourly
def run_hourly():
    print("hourly")

@once_only.Once(datetime.timedelta(minutes=60))
def run_every_60_minutes():
    print("60 minutes")

run_minutely()
run_hourly()
run_every_60_minutes()
```

will print "minutely" and "hourly", but not "60 minutes" because 60 minutes is the same as one hour, so the 60 minutes timer will already be triggered by `run_hourly` and `run_every_60_minutes` will not be run.

Further documentation can be found at: https://once_only.readthedocs.io.

1.3 License

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install *once_only*, run this command in your terminal:

```
$ pip install once_only
```

This is the preferred method to install *once_only*, as it will always install the most recent stable release.

If you don’t have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for *once_only* can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/mikapfl/once_only
```

Or download the tarball:

```
$ curl -OJL https://github.com/mikapfl/once_only/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

API

3.1 Module contents

Run a python script or function only once in a given time frame.

class once_only.**Once**(*timedelta: timedelta*, *file: Optional[Path] = None*)

Bases: `object`

Class for time keeping, enabling you to run functions only once in a given time span.

check_ready() → bool

Check if the last execution was longer ago than the timedelta.

Returns True if the last execution was longer ago, False otherwise. Does not record a new execution.

check_ready_trigger() → bool

Check if the last execution was longer ago than the timedelta, and record a new execution.

If the last execution was longer ago than the timedelta, return True and records a new execution now. Otherwise, returns False.

CHAPTER

FOUR

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/mikapfl/once_only/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Write Documentation

once_only could always use more and better documentation!

4.1.4 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/pflueger/once_only/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *once_only* for local development.

1. Fork the *once_only* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/once_only.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ cd once_only/
   $ make virtual-environment
   $ make install-pre-commit
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass our tests and automatically format everything according to our rules:

   ```
   $ make lint
   ```

   Often, the linters can fix errors themselves, so if you get failures, run `make lint` again to see if any errors need human intervention.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring and check the generated API documentation.
3. The pull request will be tested on python 3.7, 3.8, 3.9, and 3.10.

4.4 Deploying

A reminder for the maintainers on how to deploy.

1. Commit all your changes.
2. Run `tbump X.Y.Z`.
3. Wait a bit that the release on github is created.
4. Upload the release to pyPI: `make release`

CHAPTER

FIVE

CREDITS

5.1 Developers

- Mika Pflüger <mika.pflueger@pik-potsdam.de>

5.2 Libraries

This package was originally created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

SIX

CHANGELOG

6.1 1.0.0 (2022-09-16)

- Initial commit.
- once_only.Once class implemented, with direct access API and decorator functionalities.
- Add Quickstart documentation.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX