Countdoom Documentation

Release 0.2.1

René-Marc Simard

Mar 07, 2020

TABLE OF CONTENTS

Python package to fetch and digest the current Doomsday Clock world threat assessment from TheBulletin.org.

Free software released under MIT License, with source code available on GitHub, Python package distributed on PyPI, and documentation hosted on Read the Docs.

CHAPTER

ONE

FEATURES

- Fetches the current Doomsday Clock value from the Bulletin of the Atomic Scientists.
- Converts the Doomsday Clock sentence into:
  - countdown seconds `60`
  - countdown minutes `1`
  - clock `11:59`
  - time `23:59:00`
- Offers a command-line interface.
- Uses Async IO for efficient Python integration.

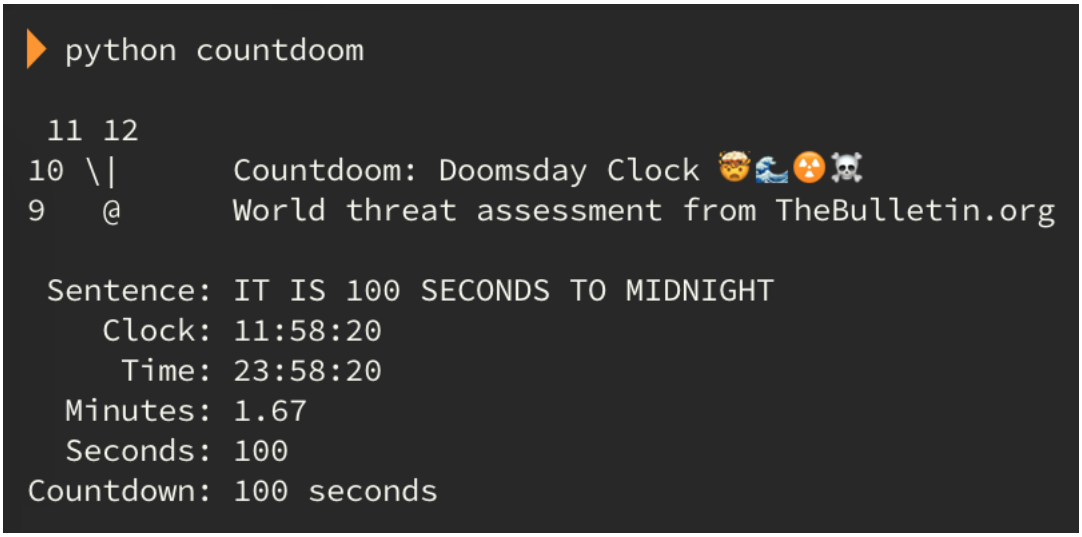

Fig. 1: *Countdoom: a Doomsday Clock client.*

1.1 Installation

1.1.1 Stable release

Countdoom is distributed on the Python Package Index (PyPI). The best way to install it is with pip:

(Optional) Create a virtual environment:

```
$ virtualenv countdoom-env
```

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

```
$ pip install countdoom
```

This is the preferred method to install **Countdoom**, 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.

1.1.2 From sources

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

You can either clone the public repository:

```
$ git clone git://github.com/renemarc/countdoom
```

Or download the tarball:

```
$ curl  -OL https://github.com/renemarc/countdoom/tarball/master
```

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

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

Or if you're on a system that supports makefiles:

```
$ make install
```

1.2 Usage

Note: The Doomsday Clock doesn't change often — at most once a year — and offers no API. Since this package relies on web scraping of TheBulletin.org, please do consider throttling/caching your requests.

1.2.1 Command-line interface

Example usage:

```
$ countdoom

 11 12
10 \|      Countdoom: Doomsday Clock
9   @      World threat assessment from TheBulletin.org

 Sentence: IT IS 2 MINUTES TO MIDNIGHT
    Clock: 11:58
     Time: 23:58:00
  Minutes: 2
  Seconds: 120
Countdown: 120 seconds
```

Example usage using a single format (e.g. `clock`):

```
$ countdoom --format clock

11:58
```

Built-in help:

```
$ countdoom -h

 11 12
10 \|      Countdoom: Doomsday Clock
9   @      World threat assessment from TheBulletin.org

usage: countdoom [--format {sentence,clock,time,minutes,countdown,all,json}]
                 [--timeout TIMEOUT] [--v] [-h]

optional arguments:
  --format {sentence,clock,time,countdown,all,json}
                        return data format (default: all).
  --timeout TIMEOUT     connection/request timeout in seconds (default: 10).
  --v, --version        show program's version number and exit
  -h, --help            show this help message and exit

"Be the change you want to see in the world." --Gandhi/Arleen Lorrance
```

1.2.2 Python import

To use **Countdoom** in a project:

```
import countdoom
```

Get the current Doomsday Clock value using the event loop:

```
1 import asyncio
2 from typing import Dict, Union
3
4 from countdoom import CountdoomClient
5
```

(continues on next page)

(continued from previous page)

```

def get_doomsday_clock() -> Dict[str, Union[str, float, None]]:
    """
    Get current Doomsday Clock value.

    :return: Dictionary of Doomsday Clock representation styles
    """
    client = CountdoomClient()
    loop = asyncio.get_event_loop()
    task = loop.create_task(client.fetch_data())
    data = loop.run_until_complete(task)
    return data
```

Get the current Doomsday Clock value using an awaitable:

```
from typing import Dict, Union

from countdoom import CountdoomClient


async def async_get_doomsday_clock() -> Dict[str, Union[str, float, None]]:
    """
    Get current Doomsday Clock value using AsyncIO.

    :return: Dictionary of Doomsday Clock representation styles
    """
    client = CountdoomClient()
    data = await client.fetch_data()
    return data
```

1.3 countdoom package

1.3.1 Submodules

1.3.2 countdoom.cli module

Console script for Countdoom.

SPDX-License-Identifier: MIT

countdoom.cli.**cli**(*args=None*)
Run Countdoom client.

Parameters **args** (Optional[List[Any]]) – list of arguments

Return type None

async countdoom.cli.**main**(*args=None*)
Console script for Countdoom.

Parameters **args** (Optional[List[Any]]) – list of arguments

Raises ***CountdoomClientError*** – If an error is generated while fetching data

Return type None

countdoom.cli.**create_parser**()
Create an argument parser.

Return type ArgumentParser

Returns Argument parser

countdoom.cli.**parse_args**(*parser*, *args*)
Feed a list of arguments into ArgumentParser for processing.

Parameters

- **parser** (ArgumentParser) – ArgumentParser instance
- **args** (list) – list of arguments

Return type Namespace

Returns ArgumentParser Namespace object

countdoom.cli.**print_header**()
Display a stylized header.

Return type None

countdoom.cli.**print_results**(*data*, *args*)
Display command results in a variety of formats.

Parameters

- **data** (dict) – command results
- **args** (Dict[str, Any]) – ArgumentParser Namespace object

Return type None

1.3.3 countdoom.client module

Client module.

SPDX-License-Identifier: MIT

class countdoom.client.**CountdoomClient**(*timeout=10*)
Bases: object

Countdoom client.

Convert Doomsday Clock data into parsable time from the Timeline page at https://thebulletin.org/doomsday-clock/past-announcements/

Based on prior Node.js work by Matt Bierner. See https://github.com/mattbierner/MinutesToMidnight

CLOCK_URL = 'https://thebulletin.org/doomsday-clock/past-announcements/'

SELECTOR = '.uabb-infobox-title'

REQUEST_TIMEOUT = 10

CLOCK_FORMAT_LONG = '%-I:%M:%S'

CLOCK_FORMAT_SHORT = '%-I:%M'

TIME_FORMAT = '%H:%M:%S'

__init__(*timeout=10*)
Create a CountdoomClient object.

Parameters **timeout** (int) – Connection/request timeout

Return type None

property countdown
Countdown to midnight.

Return type Optional[float]

Returns Number of seconds to midnight

property sentence
Doomsday Clock sentence.

Return type Optional[str]

Returns Doomsday Clock sentence

clock()
Convert countdown to midnight into a clock representation.

Return type Optional[str]

Returns Clock representation of a countdown to midnight

minutes()
Convert countdown to midnight into minutes to midnight representation.

Return type Optional[float]

Returns Number of minutes to midnight

time(*time_format='%H:%M:%S'*)
Convert countdown to midnight into a time representation.

Parameters **time_format** (str) – strftime() time format

Return type Optional[str]

Returns Time representation of a countdown to midnight

async fetch_data()
Retrieve the parsed Doomsday Clock.

Return type Dict[str, Union[str, float, None]]

Returns Extracted sentence, clock, time, minutes, and countdown

async close()
Close the HTTP connection.

Return type None

classmethod sentence_to_countdown(*sentence*)
Convert Doomsday Clock sentence to a number of seconds to midnight.

Parameters **sentence** (str) – Doomsday Clock sentence

Return type float

Returns A countdown to midnight

Raises **AttributeError** – If sentence is not matched by regex pattern

static numeric_word_to_int(*word*)
Convert textual numbers into integers.

Parameters **word** (str) – Textual number from zero to nine

Return type Optional[int]

Returns Number from 0 to 9, if any

Todo throw exception when word not found.

static countdown_to_time(*number*, *time_format='%H:%M:%S'*)
Convert a number of seconds to midnight into a time format.

Parameters

- **number** (Union[int, float]) – Number representing a countdown
- **time_format** (str) – strftime() time format

Return type str

Returns Time representation of countdown to midnight

exception countdoom.client.**CountdoomClientError**
Bases: Exception

Countdoom client general error.

1.3.4 Module contents

Top-level package for Countdoom.

SPDX-License-Identifier: MIT

1.4 Contributing

Contributions are welcome, and they are greatly appreciated! This project follows the all-contributors specification: every little bit helps and credit will always be given.

Note: This project is released with a respect oriented Contributor Code of Conduct based on the Contributor Covenant. By participating in this project you agree to abide by its fair terms.

You can contribute in many ways:

1.4.1 Types of contributions

Report bugs

Please report bugs at https://github.com/renemarc/countdoom/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.

Fix bugs

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

Implement features

Look through the GitHub issues for features. Anything tagged with `enhancement` and `help wanted` is open to whoever wants to implement it.

Write documentation

Countdoom could always use more documentation, whether as part of the official **Countdoom** docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit feedback

The best way to send feedback is to file an issue at https://github.com/renemarc/countdoom/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 this is a volunteer-driven project, and that contributions are welcome

1.4.2 Get started!

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

Note: While **Countdoom** runs on Python 3.5+, many dev tools will require Python 3.6+.

---

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

```
$ git clone git@github.com:YOUR_USERNAME_HERE/countdoom.git
```

3. Create a virtual environment for local development:

```
$ cd countdoom/
$ python -m venv .venv
$ . .venv/bin/activate
```

4. Install your local copy with all dependencies using pip:

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

Alternatively, you can also use `setup.py` to install the above requirements:

```
$ pip install --upgrade setuptools
$ python setup.py develop
```

5. Create a branch for local development:

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

Now you can make your changes locally!

6. To help you test your code, you can use pyenv version manager to install concurrent Python versions in local virtual environments (unless already installed):

```
$ pyenv install "3.5.9"
$ pyenv install "3.6.10"
$ pyenv install "3.7.6"
$ pyenv install "3.8.1"
$ pyenv install "pypy3.6-7.3.0"
$ pyenv local "3.5.9" "3.6.10" "3.7.6" "3.8.1" "pypy3.6-7.3.0"
```

7. When you're done making changes, you can test the results with makefile. This will verify that your changes pass this opinionated code quality gauntlet :

- black code formatter
- flake8 style enforcer
- isort imports checker
- mypy static type checker
- pylint code analyzer
- pytest python tests
- tox multi-version automated testing tool

```
$ make test-all
$ make coverage
```

Alternatively, you can run the test suites individually:

```
$ black --check --diff .
$ flake8
$ isort --check -rc .
$ mypy
$ pylint setup.py countdoom examples
$ pylint --disable=E0401 tests/*.py
$ pytest
$ tox -e py35
$ tox -e py36
$ tox -e py37
$ tox -e py38
$ tox -e pypy3
$ coverage
```

Note: To run a subset of tests, you can mention either the whole file or just one function:

```
$ pytest tests/test_client.py
$ pytest tests/test_client.py::test_valid_countdown
```

---

8. Commit your changes using Conventional Commits comment style and push your branch to GitHub. To help catch any gotchas, pre-commit will automatically run various code quality linters on any modified files:

```
$ git add .
$ git commit -m "type(scope): detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

9. Submit a pull request through the GitHub website.

1.4.3 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, mention the change in the `CHANGELOG.rst`, and if necessary add the feature to the list in `README.md` (repo) and `README.rst` (docs).
3. The pull request should work for Python 3.5, 3.6, 3.7, 3.8, and for PyPy3. Check https://travis-ci.com/renemarc/countdoom/pull_requests and make sure that the tests pass for all supported Python versions.

1.4.4 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in CHANGELOG.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis CI will then deploy to the Python Package Index if tests pass.

1.5 Credits

This project follows the all-contributors specification (emoji key available here). Found a bug? Want to suggest an idea? Want to share some improvements? *Contributions of any kind are welcome!*

1.6 Changelog

All notable changes to **Countdoom** will be documented in this list. The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

1.6.1 Unreleased

No documented unreleased changes

1.6.2 v0.2.1 — 2020-03-07

Added

- Code of Conduct based on Contributor Covenant.
- Continuous deployment to PyPI when new versions are pushed to the repo.

Fixed

- Spelling, links and images in documentation.

1.6.3 v0.2.0 — 2020-03-03

Alpha release. Since the Doomsday Clock has (. . . unfortunately) started counting in seconds for the first time since 1947, the code was adapted to also handle sub-minute values.

Added

- `minutes` as an output format option and in returned data set.
- Repo-specific Markdown README file.
- Documentation at Read The Docs.
- Pull Request template.
- Support for All Contributors specifications and service app.
- Support for Code Climate code quality checker service.
- Support for CodeCov test coverage report analyzer service.
- Support for DeepSource code security checker service.
- Support for Keep a Changelog specifications.
- Support for mypy static type checker.
- Support for Probot's Auto-Comment response service app.
- Support for Probot's helPR issue labeler service app.
- Support for Probot's No Response Info issue closing service app.
- Support for Probot's Request Info issue validating service app.
- Support for Probot's Stale Info auto-closing service app.
- Support for Probot's Welcome greeting service app.

Changed

- **BREAKING:** Return countdown in seconds instead of minutes.
- **BREAKING:** Rename project to **Countdoom**.
- Expand Tox test environments to include Python 3.5–3.8, Pypy3, formatter, and linters.
- Expand test coverage to cover seconds to midnight.
- Improve type hints.
- Expand contributing guidelines.
- Improve install documentation.
- Move Asyncio loop handling from package `__main__.py` to `cli.py`.
- Simplify support tools configuration files.
- Regroup dependencies listing to `setup.py`.
- Add descriptive file headers and modelines.
- Split Issue template into *Bug Report*, *Feature Request*, *Questions and Help*, and *Agile User Story*.

Fixed

- Revise sentence extraction logic to include seconds to midnight.

Removed

- Files `requirements.txt` and `requirements_dev.txt` (now in `setup.py`).
- Support for Pyup dependency checker service.

1.6.4 v0.1.0 — 2020-02-23

Inital release.

Added

- Extraction of minutes to midnight from TheBulletin.org.
- Tests with pytest.
- Command-line interface.
- Integration examples.
- Importable client module with Asyncio support.
- Makefile build assistant.
- Basic Sphinx documentation.
- Badges to README file.
- Support for bandit security issues checker.
- Support for Black code formatter.

- Support for Coverage.py unit tests measuring tool.
- Support for EditorConfig coding style config file.
- Support for Flake8 coding style enforcer.
- Support for isort imports organizer.
- Support for pip dependencies manager.
- Support for pre-commit git hooks with linters, formatters, and validators.
- Support for Pylint code analyzer.
- Support for Pyup dependency checker service.
- Support for Tox automation integration.
- Support for Travis-CI continuous integration service.

1.7 General indices

- genindex
- Index of Documented Public Modules

PYTHON MODULE INDEX

C

INDEX

Symbols

C

F

M

N

P

R

S

T