devhelpers

Release Unknown

Michael Sasser

Jun 30, 2021

CONTENTS:

DevTools (or `devhelpers`) is a loose collection of python development helpers.

CHAPTER

ONE

INSTALLATION

1.1 Instalation with pip

To install DevHelpers run `pip install devhelpers` with a `Python>=3.8`. If you already have a version of DevHelpers installed, you can upgrade it with `pip install --upgrade devhelpers`.

CHAPTER

TWO

CONTRIBUTER DOCUMENTATION

First off, thank you for considering contributing to DevHelpers. Please make sure to read our Code of Conduct before you start Contributing to DevHelpers.

2.1 Contributor Covenant Code of Conduct

2.1.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

2.1.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

2.1.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

2.1.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

2.1.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at Abuse@MichaelSasser.org. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

2.1.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

2.1.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

[homepage]: https://www.contributor-covenant.org
[v2.0]: https://www.contributor-covenant.org/version/2/0/code_of_conduct.html
[Mozilla CoC]: https://github.com/mozilla/diversity
[FAQ]: https://www.contributor-covenant.org/faq
[translations]: https://www.contributor-covenant.org/translations

2.2 I found a bug / I want to give feedback

If you found a bug or you want to give feedback, please create an issue using one of the templates.

2.3 I have a question

Please check the discussions first. When you don't find the right thread, feel free to create a new one.

2.4 Add a feature

Note: Before you start make sure you hand in an issue. Describe, what you like to change/add, so others are informed, what you are about to change and why you want to change anything.

1. Make sure you have at least `Python 3.9`, poetry, and pre-commit installed.
2. Create a fork of devhelpers.
3. Clone the fork (`origin`) to your local machine.
4. Add the original repository as a remote named `upstream`.
5. Create a new branch from the develop branch. Make sure you use the git-flow branching model scheme. (You don't necessarily need `git-flow`). Example: Let's say your issue was issue `#42` and you want to create a feature. Your branch name would be `feature/#42` or `feature/#42-my-cool-feature`.
6. Install the required tools with `poetry install -E docs`
7. Implement your feature or fix the bug you described in your issue.
8. Create a `Pull Request` as soon as possible as `draft`, so other contributors are able to help you and follow your progress.
9. Make sure to add/alter the documentation.
10. Add/alter tests, to test your code.
11. Describe your changes in one sentence in a newsfragment in `./news/`. You find the categories in the `pyproject.toml` under the `[tool.towncrier]` -> `directory`. Example: Let's say your issue was issue `#42` and you added a bugfix. Give the newsfragment the name `42.bugfix`. A feature would be called `42.feature`.
12. Run `tox`. If everything is green with no errors, you are good to go.
13. Publish your branch to your fork (`origin`).
14. Create a pull request from the Branch, which contains your changes to devhelpers's `develop` branch.
15. Once the pull request is reviewed and merged you can pull the changes from `upstream` (the original repository) to your local repository and start over again from `5.`. Don't forget to create an issue first.

Note: Do not add any additional requirement without an approval first.

Note: If you have any questions feel free to ask in the issues, pull requests and discussions.

Note: You often can use one if the *Tools* as template for a new feature.

2.4.1 Tools

TimeIt

devhelpers.timeit.**timeit**(*arg*)

Use the decorator to time functions in place.

One hazard: When you use the decorator with a function/method that does change something outside it's namespace or a method changes anything inside the internal dict, and you let it repeat stuff, might result in an unexpected behavior.

Notes

It is safe:

- when the function/method does not e.g. count, append something outside its own namespace, when using the decorator with repeating enabled like `@timeit(100)`. Or in other words: if the internal state does not change, when you call it $ n+1 $ times.
- if you don't repeat anything: `@timeit`.

Examples

```
@timeit(100)
def foo(a, b):
    pass

@timeit
def bar(c, d):
    pass
```

Parameters **arg** (Union[Callable[..., ~ReturnType], int]) – the callable or number of runs

Return type Callable[..., ~ReturnType]

IsAtomic

exception devhelpers.isatomic.**NotAtomicError**(*result_1*, *result_n*, *iteration*)

Bases: Exception

devhelpers.isatomic.**isatomic**(*arg*)

Use the decorator to check if a function is atomic.

The decorator runs a function over and over again and compares the output. When the output changes, it raises a *NotAtomicError*.

Examples

```
@isatomic(100)
def foo(a, b):
    pass

@isatomic
def bar(c, d):
    pass
```

Parameters **arg** (Union[Callable[..., ~ReturnType], int]) – the callable or number of runs

Return type Callable[..., ~ReturnType]

NoGC

devhelpers.nogc.**nogc**(*func*)

Use the decorator to disable the garbage collector for a function.

The garbage collector runs frequently to remove unreachable objects from memory. While running the `@timeit` decorator, to time functions it might be beneficial to disable the garbage collector.

Notes

By disabling the garbage collector, dangling object remain in memory until the garbage collector is re-enabled and the garbage is freed.

If you are sure, you don't have a "leaking" function, this might not be a problem. But if the program is "leaky", and you use e.g. the `timeit` decorator, each cycle new objects are created but never freed.

Make sure you understand the reference counter and the garbage collector to not use this decorator for the wrong reasons.

Examples

```
@nogc
@timeit(100)
def foo(a, b):
    pass

@nogc
@timeit
def bar(c, d):
    pass

@nogc
def baz():
    assert not gc.isenabled()
```

Return type Callable[..., ~ReturnType]

2.4.2 Tools (Tests)

TimeIt

IsAtomic

NoGC

CHAPTER

THREE

CHANGELOG

This is the changelog of DevHelpers. You can find the issue tracker on GitHub.

> **Warning:** DevHelpers is not intended to be used in a final product.

CHAPTER

FOUR

BRANCHING MODEL

This repository uses the git-flow branching model by Vincent Driessen. It has two branches with infinite lifetime:

- master
- develop

The master branch gets updated on every release. The develop branch is the merging branch.

CHAPTER

FIVE

SEMANTIC VERSIONING

After release "1.0.0" this repository will use SemVer for its release cycle.

Note: Before release "1.0.0" it uses "0.y.z" as recommanded by SemVer. This means that breaking changes result in a version change at "y" position (e.g. "0.1.0" -> "0.2.0"). Non breaking changes result in a "z" change (e.g. "0.1.1" -> "0.1.2").

CHAPTER

SIX

INDICES & TABLES

- genindex
- modindex
- search

CHAPTER

SEVEN

LICENSE

PYTHON MODULE INDEX

d

INDEX