Parsenvy

Release 3.0.2

Nik Kantar <nik@nkantar.com>

Jan 01, 2024

CONTENTS:

CHAPTER

ONE

PARSENVY: ENVIOUSLY ELEGANT ENVIRONMENT VARIABLE PARSING

Parsenvy is an *enviously* elegant environment variable parsing Python library.

Environment variables are strings by default. This can be *rather* inconvenient if you're dealing with a number of them, and in a variety of desired types. Parsenvy aims to provide an intuitive, explicit interface for retrieving these values in appropriate types with *human-friendly* syntax.

1.1 Features

- Compatible with Python 3.8+ only (the last Python 2 compatible version was 1.0.2).
- Fully tested on Linux, macOS, and Windows.
- No dependencies outside of the Python standard library.
- BSD (3-Clause) licensed.
- Utterly awesome.
- Now with docs!

1.2 Examples

```
>>> import parsenvy

>>> parsenvy.bool('DEBUG_ENABLED')  # DEBUG_ENABLED=True
True

>>> parsenvy.int('POSTS_PER_PAGE')  # POSTS_PER_PAGE=13
13

>>> parsenvy.float('EXCHANGE_RATE')  # EXCHANGE_RATE=42.911
```

(continues on next page)

(continued from previous page)

```
42.911

>>> parsenvy.list('INVALID_USERNAMES')  # INVALID_USERNAMES=admin,superuser,user,
↪webmaster
['admin', 'superuser', 'user', 'webmaster']

>>> parsenvy.tuple('SAMPLE_GREETING')  # SAMPLE_GREETING=Hello,world!
('Hello', 'world!')

>>> parsenvy.set('ALLOWED_CATEGORIES')  # ALLOWED_CATEGORIES=python,vim,git
{'python', 'vim', 'git'}

>>> parsenvy.str('DB_PREFIX')  # DB_PREFIX=dj_
'dj_'
```

1.3 Install

```
pip install parsenvy
```

1.4 Contributing

Contributions are welcome, and more information is available in the contributing guide.

1.4.1 Boolean

`parsenvy.parsenvy.`**`bool`**(*env_var: str*, *default: bool | None = None*) → bool | None

Parse environment variable value into a boolean.

Parameters

- **`env_var`** (*`str`*) – Name of desired environment variable.
- **`default`** (*`bool, optional`*) – Optional fallback value.

Returns

Environment variable typecast into a boolean.

Return type

bool (optional)

Usage

```
export DEBUG_ENABLED=True
```

```
>>> import parsenvy
>>> parsenvy.bool('DEBUG_ENABLED')
True
```

1.4.2 Integer

`parsenvy.parsenvy.`**`int`**(*env_var: str*, *default: int | None = None*) → int | None

Parse environment variable value into a integer.

Parameters

- **`env_var`** (*`str`*) – Name of desired environment variable.
- **`default`** (*`int, optional`*) – Optional fallback value.

Returns

Environment variable typecast into a integer.

Return type

int (optional)

Usage

```
export POSTS_PER_PAGE=13
```

```
>>> import parsenvy
>>> parsenvy.bool('POSTS_PER_PAGE')
13
```

1.4.3 Float

`parsenvy.parsenvy.`**`float`**(*env_var: str*, *default: float | None = None*) → float | None

Parse environment variable value into a float.

Parameters

- **`env_var`** (*`float`*) – Name of desired environment variable.
- **`default`** (*`float, optional`*) – Optional fallback value.

Returns

Environment variable typecast into a float.

Return type

float (optional)

Usage

```
export EXCHANGE_RATE=42.911
```

```
>>> import parsenvy
>>> parsenvy.float('EXCHANGE_RATE')
42.911
```

1.4.4 List

`parsenvy.parsenvy.list`(*env_var: str*, *default: List[Any] | None = None*) → List[Any] | None

Parse environment variable value into a list.

Parameters

- **env_var** (*str*) – Name of desired environment variable.
- **default** (*List, optional*) – Optional fallback value.

Returns

Environment variable typecast into a list.

Return type

List (optional)

Usage

```
export INVALID_USERNAMES=admin,superuser,user,webmaster
```

```
>>> import parsenvy
>>> parsenvy.list('INVALID_USERNAMES')
['admin', 'superuser', 'user', 'webmaster']
```

1.4.5 Tuple

`parsenvy.parsenvy.tuple`(*env_var: str*, *default: Tuple[Any, ...] | None = None*) → Tuple[Any, ...] | None

Parse environment variable value into a tuple.

Parameters

- **env_var** (*str*) – Name of desired environment variable.
- **default** (*tuple, optional*) – Optional fallback value.

Returns

Environment variable typecast into a tuple.

Return type

tuple (optional)

Usage

```
export SAMPLE_GREETING=Hello,world!
```

```
>>> import parsenvy
>>> parsenvy.tuple('SAMPLE_GREETING')
('Hello', 'world!')
```

1.4.6 Set

`parsenvy.parsenvy.`**`set`**(*env_var: str*, *default: Set[Any] | None = None*) → Set[Any] | None

Parse environment variable value into a set.

Parameters

- **`env_var`** (*`str`*) – Name of desired environment variable.
- **`default`** (*`set, optional`*) – Optional fallback value.

Returns

Environment variable typecast into a set.

Return type

set (optional)

Usage

```
export ALLOWED_CATEGORIES=python,vim,git
```

```
>>> import parsenvy
>>> parsenvy.set('ALLOWED_CATEGORIES')
{'python', 'vim', 'git'}
```

1.4.7 String

`parsenvy.parsenvy.`**`str`**(*env_var: str*, *default: str | None = None*) → str | None

Parse environment variable value into a str.

Parameters

- **`env_var`** (*`str`*) – Name of desired environment variable.
- **`default`** (*`str, optional`*) – Optional fallback value.

Returns

Environment variable typecast into a str.

Return type

str (optional)

Usage

```
export DB_PREFIX=dj_
```

```
>>> import parsenvy
>>> parsenvy.str('DB_PREFIX')
'dj_'
```

1.4.8 default_if_none

`default_if_none` is a decorator used on all Parsenvy functions as a shortcut to return the supplied default fallback if the desired environment variables isn't defined.

It isn't relevant to the users of the library—only those making changes to its code.

`parsenvy.parsenvy.`**`default_if_none`**(*func: Callable[[str, Any], Any]*) → Any | None

Decorate function to return default if desired env var isn't defined.

Parameters
`func` (*`callable`*) – Function to return if desired env var is defined.

Returns
Decorated function.

Return type
callable (optional)

1.4.9 Contributing

Contributing to Parsenvy

Hi! First of all, thank you for contributing. :heart:

All of the usual sorts of contributions are welcome: bug reports, patches, and feedback. Feel free to browse existing issues or create a new one.

Participation in this project in any way requires complying with the Code of Conduct.

Got a problem?

You're welcome to create an issue, but please search existing ones first to see if it's been discussed before.

Want to submit some code or docs?

Great! If you're intersted in tackling an existing issue, comment on one to make sure you're on the right track. If it's an idea you have or a problem not captured in an issue, create one and let's align.

Dev setup

Requirements:

- Python 3.6 or higher
- Poetry

Once you have those two, install project dependencies with:

```
poetry install
```

At this point you should be able to make changes to the codebase and run things.

Speaking of running things, there are a number of code quality checks that get run on every pull request. You are encouraged to run them locally as you work for a much faster feedback loop.

Table 1: Commands

Command	What it does
make formatcheck	Run the *black* formatter with the *–check* flag to validate the source is formatted correctly.
make lint	Run the *flake8* linter to validate there are no linter errors in the source.
make doccheck	Run the *pydocstyle* docstring format checker to validate the docstrings are complete and standardized. *Currently disabled*.
make docs	Run the Sphinx documentation site builder to validate the docs build successfully.
make typecheck	Run the *mypy* type checker to validate the type annotations are valid. *Currently disabled*.
make test	Run tests to validate they all pass.
make covcheck	Calculate test coverage to validate all code paths are executed at least once.

Docstring standards

The project uses the Google style of docstrings.

1.4.10 Code of Conduct

Contributor Covenant Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

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

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers 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, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at nik+parsenvy@nkantar.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4.

1.4.11 Changelog

Parsenvy Changelog

Unreleased

Fixed

- Mismatch between env value and resulting tuple in docs (issue #63, PR #64).

3.0.2 - 2021-02-22

Fixed

- Version number in docs

3.0.1 - 2021-02-22

Added

- All relevant files to the built wheel

Changed

- Made `parsenvy` module exports "explicit"
- Updated docs:
  - Added repo and package external links to the sidebar
  - Updated examples in the README
  - Updated examples in the docs
  - Added Contributing Guidelines, Code of Conduct, and Changelog pages and sidebar links

Fixed

- PyPI badge in README

3.0.0 - 2021-02-21

Added

- CI with code quality checks
- CD with automatic publishing to PyPI
- Contributing guidelines
- Sphinx generated docs, both locally and on Read the Docs
- Docs for *all* the things!
- Tests for *all* the things!
- ***SOME CONTRIBUTORS!!! :D***

Changed

- Refactored everything to v3
- Dev env management from Pipenv to Poetry
- Some docs from MD to rST

Removed

- `dict()`

2.1.0 - 2019-05-03

Deprecated

- `dict()`

2.0.10 - 2019-02-15

Added

- Windows testing via AppVeyor. :)

2.0.9 - 2019-02-15

Fixed

- Package name typo in setup.py.

2.0.8 - 2019-02-15

Changed

- Fixed bad URL in setup.py.

2.0.7 - 2019-02-15

Changed

- Fixed bad version in setup.py.

2.0.6 - 2019-02-15

Changed

- Fixed README type in setup.py.

2.0.5 - 2019-02-15

Changed

- Updated setup.py per most recent docs

2.0.4 - 2019-02-15

Changed

- Moved mypy and Black to dev dependencies.

Removed

- GitHub templates

2.0.3 - 2019-02-15

Added

- Black code formatter.

Changed

- Formatted all Python files with Black.

2.0.2 - 2019-02-15

Fixed

- Typing issues were causing Travis CI to fail.

Changed

- Dev environment is now managed by Pipenv.

2.0.1 - 2018-02-11

Added

- MyPy to Travis CI

2.0.0 - 2018-02-11

Added

- Type hints!

Removed

- Python 2 support!
- Also support for Python 3.2, 3.3, 3.4, 3.5 and PyPy2.7

1.0.2 - 2018-02-11

Changed

- Fixed bad version linking in the changelog.
- Added release dates to the changelog.
- Added Code of Conduct reference to the README.
- Updated Code of Conduct email.
- Updated first release link from the commit to the release tag.

Fixed

- Two of the functions returned the wrong thing instead of the default.

1.0.1 - 2017-07-31

Added

- Travis CI
- SayThanks.io badge to README
- Code of Conduct
- Contributing Guidelines
- GitHub issue/PR templates

1.0.0 - 2017-04-28

Changed

- Neater import: `from parsenvy import parsenvy` is now `import parsenvy`.

0.1.2 - 2017-04-28

Added

- Support for Python 3

Changed

- updated `README.rst`

0.1.1 - 2017-03-31

Changed

- Updated `README.rst`

0.1.0 - 2017-03-31

Added

- Basic functionality

INDEX