python-semver Documentation

Kostiantyn Rybnikov and all

Nov 05, 2019

Contents:

CHAPTER 1

Welcome to python-semver's documentation!

1.1 Installation

For Python 2:

```
pip install semver
```

For Python 3:

```
pip3 install semver
```

1.2 Usage

This module provides just couple of functions, main of which are:

```
>>> import semver
>>> semver.compare("1.0.0", "2.0.0")
-1
>>> semver.compare("2.0.0", "1.0.0")
1
>>> semver.compare("2.0.0", "2.0.0")
0
>>> semver.match("2.0.0", ">=1.0.0")
True
>>> semver.match("1.0.0", ">1.0.0")
False
>>> semver.format_version(3, 4, 5, 'pre.2', 'build.4')
'3.4.5-pre.2+build.4'
>>> version_parts = semver.parse("3.4.5-pre.2+build.4")
>>> version_parts == {
...     'major': 3, 'minor': 4, 'patch': 5,
```

(continues on next page)

(continued from previous page)

```
...     'prerelease': 'pre.2', 'build': 'build.4'}
True
>>> version_info = semver.parse_version_info("3.4.5-pre.2+build.4")
>>> # or using static method parse
>>> from semver import VersionInfo
>>> version_info = VersionInfo.parse("3.4.5-pre.2+build.4")
>>> version_info
VersionInfo(major=3, minor=4, patch=5, prerelease='pre.2', build='build.4')
>>> version_info.major
3
>>> version_info > (1, 0)
True
>>> version_info < (3, 5)
True
>>> semver.bump_major("3.4.5")
'4.0.0'
>>> semver.bump_minor("3.4.5")
'3.5.0'
>>> semver.bump_patch("3.4.5")
'3.4.6'
>>> semver.max_ver("1.0.0", "2.0.0")
'2.0.0'
>>> semver.min_ver("1.0.0", "2.0.0")
'1.0.0'
```

1.3 How to Contribute

When you make changes to the code please run the tests before pushing your code to your fork and opening a pull request:

```
python setup.py test
```

We use py.test and tox to run tests against all supported Python versions. All test dependencies are resolved automatically, apart from virtualenv, which for the moment you still may have to install manually:

```
pip install "virtualenv<14.0.0"  # <14.0.0 needed for Python 3.2 only
```

You can use the `clean` command to remove build and test files and folders:

```
python setup.py clean
```

1.4 API

Python helper for Semantic Versioning (http://semver.org/)

class semver.**VersionInfo**(*major*, *minor*, *patch*, *prerelease=None*, *build=None*)

Parameters

- **major** (*int*) – version when you make incompatible API changes.
- **minor** (*int*) – version when you add functionality in a backwards-compatible manner.
- **patch** (*int*) – version when you make backwards-compatible bug fixes.

- **prerelease** (*str*) – an optional prerelease string
- **build** (*str*) – an optional build string

build

major

minor

static parse(*version*)

Parse version string to a VersionInfo instance.

```
>>> from semver import VersionInfo
>>> VersionInfo.parse('3.4.5-pre.2+build.4')
VersionInfo(major=3, minor=4, patch=5, prerelease='pre.2', build='build.4')
```

Parameters **version** – version string

Returns a *VersionInfo* instance

Return type *VersionInfo*

patch

prerelease

semver.**bump_build**(*version*, *token='build'*)

Raise the build part of the version

Parameters

- **version** – version string
- **token** – defaults to 'build'

Returns the raised version string

Return type str

```
>>> bump_build('3.4.5-rc.1+build.9')
'3.4.5-rc.1+build.10'
```

semver.**bump_major**(*version*)

Raise the major part of the version

Param version string

Returns the raised version string

Return type str

```
>>> import semver
>>> semver.bump_major("3.4.5")
'4.0.0'
```

semver.**bump_minor**(*version*)

Raise the minor part of the version

Param version string

Returns the raised version string

Return type str

```
>>> import semver
>>> semver.bump_minor("3.4.5")
'3.5.0'
```

semver.**bump_patch**(*version*)

Raise the patch part of the version

Param version string

Returns the raised version string

Return type str

```
>>> import semver
>>> semver.bump_patch("3.4.5")
'3.4.6'
```

semver.**bump_prerelease**(*version*, *token='rc'*)

Raise the prerelease part of the version

Parameters

- **version** – version string
- **token** – defaults to 'rc'

Returns the raised version string

Return type str

```
>>> bump_prerelease('3.4.5', 'dev')
'3.4.5-dev.1'
```

semver.**cmp**(*a*, *b*)

semver.**compare**(*ver1*, *ver2*)

Compare two versions

Parameters

- **ver1** – version string 1
- **ver2** – version string 2

Returns The return value is negative if ver1 < ver2, zero if ver1 == ver2 and strictly positive if ver1 > ver2

Return type int

```
>>> import semver
>>> semver.compare("1.0.0", "2.0.0")
-1
>>> semver.compare("2.0.0", "1.0.0")
1
>>> semver.compare("2.0.0", "2.0.0")
0
```

semver.**finalize_version**(*version*)

Remove any prerelease and build metadata from the version

Parameters **version** – version string

Returns the finalized version string

Return type str

```
>>> finalize_version('1.2.3-rc.5')
'1.2.3'
```

semver.**format_version**(*major*, *minor*, *patch*, *prerelease=None*, *build=None*)

Format a version according to the Semantic Versioning specification

Parameters

- **major** (*str*) – the required major part of a version
- **minor** (*str*) – the required minor part of a version
- **patch** (*str*) – the required patch part of a version
- **prerelease** (*str*) – the optional prerelease part of a version
- **build** (*str*) – the optional build part of a version

Returns the formatted string

Return type str

```
>>> import semver
>>> semver.format_version(3, 4, 5, 'pre.2', 'build.4')
'3.4.5-pre.2+build.4'
```

semver.**match**(*version*, *match_expr*)

Compare two versions through a comparison

Parameters

- **version** (*str*) – a version string
- **match_expr** (*str*) – operator and version; valid operators are < smaller than > greater than >= greator or equal than <= smaller or equal than == equal != not equal

Returns True if the expression matches the version, otherwise False

Return type bool

```
>>> import semver
>>> semver.match("2.0.0", ">=1.0.0")
True
>>> semver.match("1.0.0", ">1.0.0")
False
```

semver.**max_ver**(*ver1*, *ver2*)

Returns the greater version of two versions

Parameters

- **ver1** – version string 1
- **ver2** – version string 2

Returns the greater version of the two

Return type *VersionInfo*

```
>>> import semver
>>> semver.max_ver("1.0.0", "2.0.0")
'2.0.0'
```

semver.**min_ver**(*ver1*, *ver2*)

Returns the smaller version of two versions

Parameters

- **ver1** – version string 1
- **ver2** – version string 2

Returns the smaller version of the two

Return type *VersionInfo*

```
>>> import semver
>>> semver.min_ver("1.0.0", "2.0.0")
'1.0.0'
```

semver.**parse**(*version*)

Parse version to major, minor, patch, pre-release, build parts.

Parameters **version** – version string

Returns dictionary with the keys 'build', 'major', 'minor', 'patch', and 'prerelease'. The prerelease or build keys can be None if not provided

Return type dict

```
>>> import semver
>>> ver = semver.parse('3.4.5-pre.2+build.4')
>>> ver['major']
3
>>> ver['minor']
4
>>> ver['patch']
5
>>> ver['prerelease']
'pre.2'
>>> ver['build']
'build.4'
```

semver.**parse_version_info**(*version*)

Parse version string to a VersionInfo instance.

Parameters **version** – version string

Returns a *VersionInfo* instance

Return type *VersionInfo*

```
>>> import semver
>>> version_info = semver.parse_version_info("3.4.5-pre.2+build.4")
>>> version_info.major
3
>>> version_info.minor
4
>>> version_info.patch
5
>>> version_info.prerelease
'pre.2'
>>> version_info.build
'build.4'
```

A Python module for semantic versioning. Simplifies comparing versions.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index