Python Bikram Samwat Documentation

Release 2.0.0

keshaB Paudel

Dec 14, 2022

Contents

Contents:

CHAPTER 1

Not actively maintained

I no longer have the time to work on this package so this is as good as abandoned.

I recommend the excellent https://github.com/dxillar/nepali-datetime package instead.

This repo and the PYPI package will still stay up but there will be no newer releases.

1.1 bikram

Utilities to work with Bikram/Vikram Samwat dates. Documentation: https://bikram.readthedocs.io.

1.2 Getting started

- Install the `bikram` package: https://bikram.readthedocs.io/installation.html.
- Read the usage guide: https://bikram.readthedocs.io/usage.html.
- Read the module reference guide: https://bikram.readthedocs.io/bikram.html.

1.3 Features

- Convert Bikram Samwat dates to AD and vice versa in a few line of codes. Intended to be useful for Nepali software developers.
- Well tested and readable source code.

- Date operations, i.e. addition/subtraction, supported with `datetime.date` and `datetime.timedelta` within range.
- Supports comparison with `datetime.date` and `datetime.timedelta` objects.
- Supports string formatting of *samwat* dates.

1.4 Caveats

- Is not very helpful if the date falls outside the map of BS years to days in month.

1.5 Credits

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

CHAPTER 2

Installation

2.1 Stable release

To install, run this command in your terminal:

```
$ pip install bikram
```

This is the recommended method of installation, as it will always install the most recent stable release.

2.2 From sources

The sources for Python Bikram Samwat can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/poudel/bikram
```

CHAPTER 3

Usage

This package contains a class named `samwat`. This is our BS date object, similar to python's native `datetime.date` object. It supports date operations with the python's native date objects.

More often than not, there is a need to work with BS and AD date at the same time. Converting back and forth to do calculation and representation becomes very tedious and results in a spagetti codebase. The `samwat` object tries to make this a bit cleaner and intuitive. Here are some examples:

3.1 samwat

Let's get today's date in Bikram Samwat.

```
>>> from datetime import date
>>> from bikram import samwat
>>> bs_date = samwat.from_ad(date.today())
>>> bs_date
samwat(2074, 11, 30)
```

Now, let's convert the `bs_date` into AD.

```
>>> bs_date.ad
datetime.date(2018, 3, 14)
```

That's it. The `samwat` instance has a property called `ad` that returns a corresponding `datetime.date` instance.

3.1.1 Out of range dates

A `ValueError` is thrown if the date you are trying to convert falls out of the range. If you are using this library to convert user-submitted dates then you need to handle this exception accordingly to avoid runtime errors.

```
>>> samwat.from_ad(date(2100, 1,1))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/danse/projects/k/bikram/bikram/bikram.py", line 113, in from_ad
    return convert_ad_to_bs(ad_date)
  File "/home/danse/projects/k/bikram/bikram/bikram.py", line 126, in convert_ad_to_bs
    raise ValueError('A.D. year is out of range...')
ValueError: A.D. year is out of range...
```

3.1.2 More

For in-depth usage and examples, go to the next page.

CHAPTER 4

bikram package

4.1 Submodules

4.2 bikram.bikram module

This module contains the `samwat`, a container class for Bikram Samwat dates.

To run the examples in this page, import `samwat` like this:

```
>>> from bikram import samwat
```

Some examples require the `datetime.date`, and `datetime.timedelta` objects. Please import them as follows:

```
>>> from datetime import date, timedelta
```

class bikram.bikram.**samwat**(*year*, *month*, *day*, *ad=None*)

Bases: `object`

This class represents a Bikram Samwat date. It can be used as an independent container, without using the date conversion part.

```
>>> samwat(2074, 11, 30)
samwat(2074, 11, 30)
```

If you have the equivalent `datetime.date` instance, then you can pass it as `_ad` argument to the constructor like this:

```
>>> samwat(2074, 11, 30, date(2018, 3, 14))
```

Doing so will cache the AD equivalent of the `samwat` instance and provide a faster access through the `ad` property for future access.

samwat also supports date operations, comparison etc. with other samwat objects and datetime.date objects. It also supports arithmetic operations with datetime.timedelta objects.

Compare two samwat date:

```
>>> samwat(2074, 10, 30) < samwat(2074, 11, 30)
True
```

Comparison with datetime.date object:

```
>>> samwat(2074, 10, 30) == date(2018, 3, 14)
True
```

Subtract 10 days from a samwat using datetime.timedelta object.

```
>>> samwat(2074, 10, 30) - timedelta(days=10)
samwat(2074, 10, 20)
```

Subtract two samwat dates and get datetime.timedelta representation.

```
>>> samwat(2074, 10, 11) - samwat(2070, 10, 11)
datetime.timedelta(1461)
```

Please note that the above operations require that the date be in the range of years specified in the constants.py file. As warned in the usage guide, you will need to handle ValueError exception if the date falls outside the range.

ad

Return a datetime.date instance, that is, this date converted to AD. Accessing the ad property automatically tries to calculate the AD date.

It caches the datetiem.date object as _ad to avoid expensive calculation for the next time.

```
>>> samwat(2074, 11, 30).ad
datetime.date(2018, 3, 14)
```

as_tuple()

Return a samwat instance as a tuple of year, month, and day.

```
>>> samwat(2074, 11, 30).as_tuple()
(2074, 11, 30)
```

day

static from_ad(*ad_date*)

Expects a *datetime.date* then returns an equivalent *bikram.samwat* instance

classmethod from_iso(*datestr: str*)

Naive way to parse date from a ISO8601 (YYYY-MM-DD) BS date string and return *bikram.samwat* instance.

month

classmethod parse(*datestr: str*, *parsestr: str*)

parse bikram samwat date string and return a *bikram.samwat* instance.

- "%d": zero padded day of month, 07
- "%-d": padded day of month, 7
- "%dne": zero-padded day of month in devanagari digits,

- "%-dne": day of month in devanagari digits,
- "%m": zero-padded month number, 01
- "%-m": month number, 1
- "%mne": zero-added month number in devanagari digits,
- "%-mne": month number in devanagari digits,
- "%y": two digit year, 73 implies 2073
- "%Y": four digit year, 2073
- "%yne": two digit year in devanagari digits, implies
- "%Yne": four digit year in devanagari digits,
- **"%B": name of bikram samwat months in English spelling, English spelling short** (abbr. by first three letters), Devanagari spelling. Any one of the list below:

"""

['', '', '', '', '', '', '', '', '', '', '', '', '',

'Baisakh', 'Jestha', 'Ashadh', 'Shrawan', 'Bhadra', 'Ashwin', 'Kartik', 'Mangsir', 'Poush', 'Magh', 'Falgun', 'Chaitra',

'Bai', 'Jes', 'Ash', 'Shr', 'Bha', 'Ash', 'Kar', 'Man', 'Pou', 'Mag', 'Fal', 'Cha',

]

"""

replace(*year=None*, *month=None*, *day=None*)

Return a new copy of `samwat` by replacing one or more provided attributes of this date. For example, to replace the year:

```
>>> samwat(2074, 11, 30).replace(year=2073)
samwat(2073, 11, 30)
```

To replace the month:

```
>>> samwat(2074, 11, 30).replace(month=12)
samwat(2074, 12, 30)
```

strftime(*formatstr: str*)

Format a samwat object to specified date string. The format strings are similar to those accepted by `parse()` with the following additions/modifications:

- "%B": Formats to Nepali month name(Example: Baisakh, Jestha, etc.)
- "%S": Formats to Punjabi Shahmukhi month name(Example: , , etc.)
- "%Bne": Formats to Nepali Devnagari month name(Example:'', '', etc.)

static today()

Returns a `samwat` instance for today.

year

bikram.bikram.**convert_ad_to_bs**(*date_in_ad*)

A function to convert AD dates to BS. Expects a *datetime.date* instance and returns an equivalent *bikram.samwat* instance.

```
>>> convert_ad_to_bs(date(2018, 3, 14))
samwat(2074, 11, 30)
```

bikram.bikram.**convert_bs_to_ad**(*date_in_bs*)

A function to convert BS dates to AD. Expects a *bikram.samwat* instance and returns an equivalent *datetime.date* instance

```
>>> convert_bs_to_ad(samwat(2074, 11, 30))
datetime.date(2018, 3, 14)
```

4.3 bikram.constants module

This file has the necessary constants for date conversion.

4.4 Module contents

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/poudel/bikram/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

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

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/poudel/bikram/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 :)

5.2 Get Started!

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

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

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

```
$ mkvirtualenv bikram
$ cd bikram/
$ python setup.py develop
```

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 flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 bikram tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

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.

5.3 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 add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, and 3.5, and for PyPy. Check https://travis-ci.org/poudel/bikram/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_bikram
```

CHAPTER 6

History

6.1 1.0.0 (2018-03-15)

- Added usage guide and module docs.
- Added `pipenv` support.
- Removed `tox`.
- Removed support for python versions older than 3.6.

6.2 0.1.2 (2016-11-13)

- Minor test fixes.

6.3 0.1.0 (2016-11-13)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index