fiscalyear Documentation

Release 0.3.2

Adam J. Stewart

Oct 20, 2021

User Documentation

fiscalyear is a small, lightweight Python module providing helpful utilities for managing the fiscal calendar. It is designed as an extension of the built-in datetime and calendar modules, adding the ability to query the fiscal year, fiscal quarter, fiscal month, and fiscal day of a date or datetime object.

CHAPTER 1

Basic Usage

`fiscalyear` provides several useful classes.

1.1 FiscalYear

The `FiscalYear` class provides an object for storing information about the start and end of a particular fiscal year.

```
>>> from fiscalyear import *
>>> a = FiscalYear(2017)
>>> a.start
FiscalDateTime(2016, 10, 1, 0, 0)
>>> a.end
FiscalDateTime(2017, 9, 30, 23, 59, 59)
>>> a.isleap
False
```

You can also get the current `FiscalYear` with:

```
>>> FiscalYear.current()
FiscalYear(2018)
```

1.2 FiscalQuarter

The `FiscalYear` class also allows you to query information about a specific fiscal quarter.

```
>>> a.q3.start
FiscalDateTime(2017, 4, 1, 0, 0)
>>> a.q3.end
FiscalDateTime(2017, 6, 30, 23, 59, 59)
```

These objects represent the standalone `FiscalQuarter` class.

```
>>> b = FiscalQuarter(2017, 3)
>>> b.start
FiscalDateTime(2017, 4, 1, 0, 0)
>>> b.end
FiscalDateTime(2017, 6, 30, 23, 59, 59)
>>> a.q3 == b
True
>>> b in a
True
>>> b.next_fiscal_quarter
FiscalQuarter(2017, 4)
```

You can also get the current `FiscalQuarter` with:

```
>>> FiscalQuarter.current()
FiscalQuarter(2018, 2)
```

1.3 FiscalMonth

The `FiscalMonth` class allows you to keep track of the fiscal month.

```
>>> c = FiscalMonth(2017, 9)
>>> c.start
FiscalDateTime(2017, 6, 1, 0, 0)
>>> c.end
FiscalDateTime(2017, 6, 30, 23, 59, 59)
>>> c in a
True
>>> c in b
True
>>> c.next_fiscal_month
FiscalMonth(2017, 10)
```

You can also get the current `FiscalMonth` with:

```
>>> FiscalMonth.current()
FiscalMonth(2018, 4)
```

1.4 FiscalDay

To keep track of the fiscal day, use the `FiscalDay` class.

```
>>> d = FiscalDay(2017, 250)
>>> d.start
FiscalDateTime(2017, 6, 6, 0, 0)
>>> d.end
FiscalDateTime(2017, 6, 6, 23, 59, 59)
>>> d in a
True
>>> d in b
True
>>> d in c
```

(continues on next page)

(continued from previous page)

```
True
>>> d.next_fiscal_day
FiscalDay(2017, 251)
```

You can also get the current `FiscalDay` with:

```
>>> FiscalDay.current()
FiscalDay(2018, 94)
```

1.5 FiscalDateTime

The start and end of each of the above objects are stored as instances of the `FiscalDateTime` class. This class provides all of the same features as the `datetime` class, with the addition of the ability to query the fiscal year, fiscal quarter, fiscal month, and fiscal day.

```
>>> e = FiscalDateTime.now()
>>> e
FiscalDateTime(2017, 4, 8, 20, 30, 31, 105323)
>>> e.fiscal_year
2017
>>> e.fiscal_quarter
3
>>> e.next_fiscal_quarter
FiscalQuarter(2017, 4)
>>> e.fiscal_month
7
>>> e.fiscal_day
190
```

1.6 FiscalDate

If you don't care about the time component of the `FiscalDateTime` class, the `FiscalDate` class is right for you.

```
>>> f = FiscalDate.today()
>>> f
FiscalDate(2017, 4, 8)
>>> f.fiscal_year
2017
>>> f.prev_fiscal_year
FiscalYear(2016)
```

CHAPTER 2

Advanced Usage

Not every government, business, and institution uses the same fiscal calendar. By default, the `fiscalyear` module uses the fiscal calendar of the U.S. federal government. For example, the 2017 fiscal year starts on October 1st, 2016 and ends on September 30th, 2017. In contrast, the United Kingdom fiscal year is completely different. For personal tax purposes, the 2017 fiscal year in the U.K. starts on April 6th, 2017 and ends on April 5th, 2018. The `fiscalyear` module allows you to change the start date of the fiscal year to suit your needs.

2.1 Changing the fiscal calendar

In order to explain how to change the fiscal calendar, let's use the U.K. personal tax financial year as an example.

```
>>> import fiscalyear
>>> a = fiscalyear.FiscalYear(2017)
```

2.1.1 Start year

The first difference you'll notice between the U.S. and the U.K. fiscal calendars is that in the U.S., the 2017 fiscal year actually starts in 2016, while in the U.K. it starts in 2017. To control this, change the start year from `'previous'` to `'same'`.

```
>>> a.start.year
2016
>>> fiscalyear.setup_fiscal_calendar(start_year='same')
>>> a.start.year
2017
```

Now that the start year is right, let's change the start month.

2.1.2 Start month

The start month can be any valid month.

```
>>> a.start.month
10
>>> fiscalyear.setup_fiscal_calendar(start_month=4)
>>> a.start.month
4
```

Finally, let's change the start day.

2.1.3 Start day

The start day can be any valid day in the chosen month.

```
>>> a.start.day
1
>>> fiscalyear.setup_fiscal_calendar(start_day=6)
>>> a.start.day
6
```

Putting everything together, we can see that the definition of the start and end of the fiscal calendar has been globally changed for all objects.

```
>>> a.start
FiscalDateTime(2017, 4, 6, 0, 0)
>>> a.end
FiscalDateTime(2018, 4, 5, 23, 59, 59)
```

Of course, we can change all of these variables at the same time like so:

```
>>> fiscalyear.setup_fiscal_calendar('same', 4, 6)
```

2.2 Temporarily changing the fiscal calendar

If you need to work with multiple fiscal calendars in the same program, it may be beneficial to be able to temporarily change the fiscal calendar. The `fiscalyear` module provides a `fiscal_calendar` context manager to handle this.

```
>>> from fiscalyear import *
>>> a = FiscalYear(2017)
>>> a.start
FiscalDateTime(2016, 10, 1, 0, 0)
>>> with fiscal_calendar(start_month=6):
...     a.start
...
FiscalDateTime(2016, 6, 1, 0, 0)
>>> a.start
FiscalDateTime(2016, 10, 1, 0, 0)
```

To recreate our previous example, this would look like:

```
>>> with fiscal_calendar('same', 4, 6):
...     a.start
...
FiscalDateTime(2017, 4, 6, 0, 0)
```

Or in a for-loop:

```
calendars = [
    ('previous', 10, 1),
    ('same', 4, 6),
    ...
]

for calendar in calendars:
    with fiscal_calendar(*calendar):
        # do stuff
```

CHAPTER 3

Installation

`fiscalyear` has no dependencies, making it simple and easy to install. There are multiple ways to install the `fiscalyear` module.

3.1 pip

The recommended way to install `fiscalyear` is with `pip`.

```
$ pip install fiscalyear
```

The fiscalyear module will now appear with your base Python installation.

3.2 Anaconda

You can also install `fiscalyear` with the `conda` package manager.

```
$ conda install -c conda-forge fiscalyear
```

3.3 Spack

If you work in an HPC environment, the Spack package manager can be used to install `fiscalyear` as well.

```
$ spack install py-fiscalyear
$ spack load py-fiscalyear
```

See the Spack Documentation to get started.

3.4 Source

If you're up for it, you can also install `fiscalyear` from source. Simply clone the repository.

```
$ git clone https://github.com/adamjstewart/fiscalyear.git
```

Now build the module.

```
$ python setup.py build
```

Finally, install the package to somewhere in your `PYTHONPATH`.

```
$ python setup.py install --prefix=/path/to/installation/prefix
```

3.5 Drag-n-Drop

If you want to vendor `fiscalyear` with your Python package, you can always download the source code. `fiscalyear` is composed of a single file, making it easy to drag-n-drop to your current directory and import.

CHAPTER 4

Testing

`fiscalyear` comes with a full test-suite called `test_fiscalyear`. To run the test-suite, you will need to install the following packages:

- pytest
- pytest-mock

4.1 Running tests

Once `pytest` is installed, simply `cd` to the root directory of `fiscalyear` and run the `pytest` command.

```
$ git clone https://github.com/adamjstewart/fiscalyear.git
$ cd fiscalyear
$ pytest
============================= test session starts ==============================
platform darwin -- Python 2.7.13, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
rootdir: /Users/Adam/fiscalyear, inifile:
plugins: cov-2.3.1
collected 66 items

test_fiscalyear.py ..................................................................

========================== 66 passed in 0.21 seconds ===========================
```

`pytest` provides automatic test detection that locates the `test_fiscalyear.py` file and runs tests that begin with `test_*`.

4.2 Continuous Integration (CI)

In order to prevent bugs from being introduced into the code, `fiscalyear` uses GitHub Actions for continuous integration. After every commit or pull request, GitHub Actions automatically runs the test-suite across all supported

versions of Python 2 and 3. This has the added benefit of preventing incompatibilities between different Python versions.

CHAPTER 5

Coverage

`fiscalyear` uses `pytest-cov` to determine the percentage of the code base that is covered by unit tests. You will need to install the following packages to run the coverage tests yourself.

- pytest
- pytest-cov
- pytest-mock

5.1 Running coverage tests

Once you have the dependencies installed, you can run the coverage tests locally.

```
$ pytest --cov=fiscalyear
============================= test session starts ==============================
platform darwin -- Python 2.7.13, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
rootdir: /Users/Adam/fiscalyear, inifile:
plugins: cov-2.3.1
collected 66 items

test_fiscalyear.py ..................................................................

---------- coverage: platform darwin, python 2.7.13-final-0 ----------
Name            Stmts   Miss  Cover
-----------------------------------
fiscalyear.py     233      0   100%


========================== 66 passed in 0.21 seconds ===========================
```

`fiscalyear` always strives for 100% coverage, and won't accept pull requests that aren't covered by unit tests.

5.2 Continuous Integration (CI)

In addition to unit tests, GitHub Actions also runs coverage tests. The results of these tests are reported back to codecov and displayed through a badge on the README.

CHAPTER 6

Documentation

`fiscalyear` uses Sphinx and the Read the Docs Theme to generate its documentation. The documentation is proudly hosted on Read the Docs.

6.1 Building the documentation

To build the documentation, you'll need to have `sphinx` and `sphinx_rtd_theme` installed. Then, `cd` to the `docs` directory and use the `Makefile` to build everything.

```
$ cd docs
$ make html
```

6.2 Building the documentation with setuptools

Alternatively, the documentation can be built with the following command:

```
$ python setup.py build_sphinx
```

6.3 Continuous Integration (CI)

Every time a change is made to the documentation and pushed to GitHub, Read the Docs automatically rebuilds the documentation, keeping everything in sync and up-to-date.

CHAPTER 7

API Docs

Utilities for managing the fiscal calendar.

class fiscalyear.**FiscalDate**

A wrapper around the builtin datetime.date class that provides the following attributes.

ctime()

Return ctime() style string.

day

fiscal_day

Returns The fiscal day

Return type int

fiscal_month

Returns The fiscal month

Return type int

fiscal_quarter

Returns The fiscal quarter

Return type int

fiscal_year

Returns The fiscal year

Return type int

fromordinal()

int -> date corresponding to a proleptic Gregorian ordinal.

fromtimestamp()

timestamp -> local date from a POSIX timestamp (like time.time()).

isocalendar()
Return a 3-tuple containing ISO year, week number, and weekday.

isoformat()
Return string in ISO 8601 format, YYYY-MM-DD.

isoweekday()
Return the day of the week represented by the date. Monday == 1 ... Sunday == 7

max = datetime.date(9999, 12, 31)

min = datetime.date(1, 1, 1)

month

next_fiscal_day

Returns The next fiscal day

Return type *FiscalDay*

next_fiscal_month

Returns The next fiscal month

Return type *FiscalMonth*

next_fiscal_quarter

Returns The next fiscal quarter

Return type *FiscalQuarter*

next_fiscal_year

Returns The next fiscal year

Return type *FiscalYear*

prev_fiscal_day

Returns The previous fiscal day

Return type *FiscalDay*

prev_fiscal_month

Returns The previous fiscal month

Return type *FiscalMonth*

prev_fiscal_quarter

Returns The previous fiscal quarter

Return type *FiscalQuarter*

prev_fiscal_year

Returns The previous fiscal year

Return type *FiscalYear*

replace()
Return date with new specified fields.

resolution = datetime.timedelta(1)

strftime()
format -> strftime() style string.

timetuple()
Return time tuple, compatible with time.localtime().

today()
Current date or datetime: same as self.__class__.fromtimestamp(time.time()).

toordinal()
Return proleptic Gregorian ordinal. January 1 of year 1 is day 1.

weekday()
Return the day of the week represented by the date. Monday == 0 . . . Sunday == 6

year

class fiscalyear.**FiscalDateTime**
A wrapper around the builtin datetime.datetime class that provides the following attributes.

astimezone()
tz -> convert to local time in new timezone tz

combine()
date, time -> datetime with same date and time fields

ctime()
Return ctime() style string.

date()
Return date object with same year, month and day.

day

dst()
Return self.tzinfo.dst(self).

fiscal_day

Returns The fiscal day

Return type int

fiscal_month

Returns The fiscal month

Return type int

fiscal_quarter

Returns The fiscal quarter

Return type int

fiscal_year

Returns The fiscal year

Return type int

fromordinal()
int -> date corresponding to a proleptic Gregorian ordinal.

fromtimestamp()
timestamp[, tz] -> tz's local time from POSIX timestamp.

hour

isocalendar()
Return a 3-tuple containing ISO year, week number, and weekday.

isoformat()
[sep] -> string in ISO 8601 format, YYYY-MM-DDTHH:MM:SS[.mmmmmm][+HH:MM].

sep is used to separate the year from the time, and defaults to 'T'.

isoweekday()
Return the day of the week represented by the date. Monday == 1 ... Sunday == 7

max = datetime.datetime(9999, 12, 31, 23, 59, 59, 999999)

microsecond

min = datetime.datetime(1, 1, 1, 0, 0)

minute

month

next_fiscal_day

Returns The next fiscal day

Return type *FiscalDay*

next_fiscal_month

Returns The next fiscal month

Return type *FiscalMonth*

next_fiscal_quarter

Returns The next fiscal quarter

Return type *FiscalQuarter*

next_fiscal_year

Returns The next fiscal year

Return type *FiscalYear*

now()
[tz] -> new datetime with tz's local day and time.

prev_fiscal_day

Returns The previous fiscal day

Return type *FiscalDay*

prev_fiscal_month

Returns The previous fiscal month

Return type *FiscalMonth*

prev_fiscal_quarter

Returns The previous fiscal quarter

Return type *FiscalQuarter*

prev_fiscal_year

Returns The previous fiscal year

Return type *FiscalYear*

replace()
Return datetime with new specified fields.

resolution = datetime.timedelta(0, 0, 1)

second

strftime()
format -> strftime() style string.

strptime()
string, format -> new datetime parsed from a string (like time.strptime()).

time()
Return time object with same time but with tzinfo=None.

timetuple()
Return time tuple, compatible with time.localtime().

timetz()
Return time object with same time and tzinfo.

today()
Current date or datetime: same as self.__class__.fromtimestamp(time.time()).

toordinal()
Return proleptic Gregorian ordinal. January 1 of year 1 is day 1.

tzinfo

tzname()
Return self.tzinfo.tzname(self).

utcfromtimestamp()
timestamp -> UTC datetime from a POSIX timestamp (like time.time()).

utcnow()
Return a new datetime representing UTC day and time.

utcoffset()
Return self.tzinfo.utcoffset(self).

utctimetuple()
Return UTC time tuple, compatible with time.localtime().

weekday()
Return the day of the week represented by the date. Monday == 0 . . . Sunday == 6

year

class fiscalyear.**FiscalDay**
A class representing a single fiscal day.

Constructor.

Parameters

- **fiscal_year** (*int or str*) – The fiscal year
- **fiscal_day** (*int or str*) – The fiscal day

Returns A newly constructed FiscalDay object

Return type *FiscalDay*

Raises

- **TypeError** – If fiscal_year or fiscal_day is not an int or int-like string
- **ValueError** – If fiscal_year or fiscal_day is out of range

classmethod current()
Alternative constructor. Returns the current FiscalDay.

Returns A newly constructed FiscalDay object

Return type *FiscalDay*

end

Returns End of the fiscal day

Return type *FiscalDateTime*

fiscal_day

Returns The fiscal day

Return type int

fiscal_month

Returns The fiscal month

Return type int

fiscal_quarter

Returns The fiscal quarter

Return type int

fiscal_year

Returns The fiscal year

Return type int

next_fiscal_day

Returns The next fiscal day

Return type *FiscalDay*

prev_fiscal_day

Returns The previous fiscal day

Return type *FiscalDay*

start

Returns Start of the fiscal day

Return type *FiscalDateTime*

class fiscalyear.**FiscalMonth**
A class representing a single fiscal month.

Constructor.

Parameters

- **fiscal_year** (*int or str*) – The fiscal year
- **fiscal_month** (*int or str*) – The fiscal month

Returns A newly constructed FiscalMonth object

Return type *FiscalMonth*

Raises

- **TypeError** – If fiscal_year or fiscal_month is not an int or int-like string
- **ValueError** – If fiscal_year or fiscal_month is out of range

classmethod current()
Alternative constructor. Returns the current FiscalMonth.

Returns A newly constructed FiscalMonth object

Return type *FiscalMonth*

end

Returns End of the fiscal month

Return type *FiscalDateTime*

fiscal_month

Returns The fiscal month

Return type int

fiscal_year

Returns The fiscal year

Return type int

next_fiscal_month

Returns The next fiscal month

Return type *FiscalMonth*

prev_fiscal_month

Returns The previous fiscal month

Return type *FiscalMonth*

start

Returns Start of the fiscal month

Return type *FiscalDateTime*

class fiscalyear.**FiscalQuarter**
A class representing a single fiscal quarter.

Constructor.

Parameters

- **fiscal_year** (*int or str*) – The fiscal year
- **fiscal_quarter** (*int or str*) – The fiscal quarter

Returns A newly constructed FiscalQuarter object

Return type *FiscalQuarter*

Raises

- **TypeError** – If fiscal_year or fiscal_quarter is not an int or int-like string

- **ValueError** – If fiscal_year or fiscal_quarter is out of range

classmethod current()
Alternative constructor. Returns the current FiscalQuarter.

Returns A newly constructed FiscalQuarter object

Return type *FiscalQuarter*

end

Returns The end of the fiscal quarter

Return type *FiscalDateTime*

fiscal_quarter

Returns The fiscal quarter

Return type int

fiscal_year

Returns The fiscal year

Return type int

next_fiscal_quarter

Returns The next fiscal quarter

Return type *FiscalQuarter*

prev_fiscal_quarter

Returns The previous fiscal quarter

Return type *FiscalQuarter*

start

Returns The start of the fiscal quarter

Return type *FiscalDateTime*

class fiscalyear.**FiscalYear**
A class representing a single fiscal year.

Constructor.

Parameters **fiscal_year** (*int or str*) – The fiscal year

Returns A newly constructed FiscalYear object

Return type *FiscalYear*

Raises

- **TypeError** – If fiscal_year is not an int or int-like string
- **ValueError** – If fiscal_year is out of range

classmethod current()
Alternative constructor. Returns the current FiscalYear.

Returns A newly constructed FiscalYear object

Return type *FiscalYear*

end

Returns End of the fiscal year

Return type *FiscalDateTime*

fiscal_year

Returns The fiscal year

Return type int

isleap
returns: True if the fiscal year contains a leap day, else False :rtype: bool

next_fiscal_year

Returns The next fiscal year

Return type *FiscalYear*

prev_fiscal_year

Returns The previous fiscal year

Return type *FiscalYear*

q1

Returns The first quarter of the fiscal year

Return type *FiscalQuarter*

q2

Returns The second quarter of the fiscal year

Return type *FiscalQuarter*

q3

Returns The third quarter of the fiscal year

Return type *FiscalQuarter*

q4

Returns The fourth quarter of the fiscal year

Return type *FiscalQuarter*

start

Returns Start of the fiscal year

Return type *FiscalDateTime*

fiscalyear.**fiscal_calendar**(**args*, ***kwds*)
A context manager that lets you modify the start of the fiscal calendar inside the scope of a with-statement.

Parameters

- **start_year** (*str*) – Relationship between the start of the fiscal year and the calendar year. Possible values: `'previous'` or `'same'`.
- **start_month** (*int or str*) – The first month of the fiscal year
- **start_day** (*int or str*) – The first day of the first month of the fiscal year

Raises

- **ValueError** – If `start_year` is not `'previous'` or `'same'`

- **TypeError** – If `start_month` or `start_day` is not an int or int-like string
- **ValueError** – If `start_month` or `start_day` is out of range

`fiscalyear.`**setup_fiscal_calendar**(*start_year=None*, *start_month=None*, *start_day=None*)

Modify the start of the fiscal calendar.

Parameters

- **start_year** (*str*) – Relationship between the start of the fiscal year and the calendar year. Possible values: `'previous'` or `'same'`.
- **start_month** (*int or str*) – The first month of the fiscal year
- **start_day** (*int or str*) – The first day of the first month of the fiscal year

Raises

- **ValueError** – If `start_year` is not `'previous'` or `'same'`
- **TypeError** – If `start_month` or `start_day` is not an int or int-like string
- **ValueError** – If `start_month` or `start_day` is out of range

Python Module Index

f

Index

N

P

Q

R

S

T

U

W

Y