undated

Release 1.0.8

rikfair

Oct 11, 2022

UNDATED

undated is a Python package that simplifies the process of working with dates that are stored as numbers or strings.

Whilst datatypes should fit their purpose, date types do not always transport well and are converted to a number or string, such as ISO 8601, for transportation in text based files like json or csv.

***undated* provides:**

- Functionality to handle dates that are stored in number formats
- Ability to derive the date format from provided data
- Direct manipulation to avoid costly type conversions

Note: Although *undated* is whizzy with dates, operating with time elements is out of its scope.

CHAPTER

ONE

INCEPTION

undated came about due to the need to process large amounts of data from various sources. These were received as csv files where the format of the date was dependant on either the country of origin, source system, or which ever way the wind blew. One solution was required to enable big data to be processed regardless of its date formatting… *undated*.

CHAPTER

TWO

CONCEPT

***undated* is a lightweight performance tuned package envisaged to be used either:**

- when processing data from various sources where the date format is unknown but consistent throughout the data
- or when dates have been stored as integers in the `Ymd` format and performance is a consideration.

For other scenarios consider using a feature rich package such as dateutils

CHAPTER

THREE

SUPPORTED VERSIONS

Supported on Python 3.7, 3.8, 3.9 and 3.10

CHAPTER

FOUR

INSTALLATION

The package can be found on GitHub and PyPI, so naturally it can be installed with *pip*

```
pip install undated
```

CHAPTER

FIVE

REQUIREMENTS

The `undated` package itself has no requirements.

To use the unittests and timings modules `dateutils` is required.

```
pip install python-dateutil>=2.8.2
```

To generate the Sphinx documentation, Sphinx and the RTD template are required.

```
pip install Sphinx>=5.0.2
pip install sphinx-rtd-theme>=1.0.0
```

CHAPTER

SIX

LICENCE

This software is released under the MIT licence

6.1 Tutorial

6.1.1 The Three Modules

undated consists of three main modules, `undated`, `undated.fmts` and `undated.utils`. These are distinctly separated for performance, to limit import overhead.

undated

The undated module, is the main module for manipulating dates, it consists of a main class object for managing a date as an integer and several functions for further functionality. All of these are hopefully self-explanatory.

undated.fmts

The `fmts` (formattings) module comes into play when the date data format is unknown. It has two key class objects. The `Deriver` class derives the date format from a list of dates, returning an `UndatedFormat` class object, which contains the information required by the `as_parts` function to extract the date elements.

undated.utils

The utils module is tuned for performance. It has little to no validation on the parameters and works solely with dates stored as integers in the `Ymd` format

---

Important: The tutorial assumes that the *undated*, *undated.fmts*, and *undated.utils* packages have been imported as *ud*, *udf* and *udu* respectfully.

```
import undated as ud
import undated.fmts as udf
import undated.utils as udu
```

6.1.2 Limitations

undated was designed with data processing of recent (-ish) dates in mind. Because of this it uses the Gregorian calendar and makes no adjustment for the Julian calendar changeover. Thus, it will treat any dates before 1583 as invalid.

6.1.3 Common Parameter Names

Parameter names have been standardised as much as possible, to be more consistent, intuitive and understandable.

- **day**: the day of the month as an integer
- **fmt**: the date format string or can sometimes be the udf.UndatedFormat object
- **idate**: a date integer of no specific format
- **iymd**: an integer in the year month day format
- **month**: the month as an integer, January == 1
- **sdate**: a date string or integer of no specific format
- **year**: the year as an integer
- **ymd**: a ud.YMD class object
- **yy_pivot**: for processing two-digit years, the lower bound for converting two-digit years to four-digit years

6.1.4 undated YMD Class

The class ud.YMD is the go-to tool, when manipulation of the dates is required during processing.

```
import undated as ud
ymd = ud.YMD(2022_07_04)
print('ymd =', ymd)
print('plus 7 days =', ymd + 7)
print('minus 7 days =', ymd - 7)
print('add 2 months =', ymd.add_months(2))
print('minus 2 months =', ymd.add_months(-2))
print('add 10 weekdays =', ymd.add_weekdays(10))
print('minus 10 weekdays =', ymd.add_weekdays(-10))
print('add 3 years =', ymd.add_years(3))
print('minus 3 years =', ymd.add_years(-3))
print('day of the week =', ymd.day_of_week())
print('is leap year =', ymd.is_leap_year())
print('is weekday =', ymd.is_weekday())
print('the day, month and year =', ymd.day, ymd.month, ymd.year)
```

gives the results

```
ymd = 20220704
plus 7 days = 20220711
minus 7 days = 20220627
add 2 months = 20220904
minus 2 months = 20220504
add 10 weekdays = 20220718
```

(continues on next page)

(continued from previous page)

```
minus 10 weekdays = 20220620
add 3 years = 20250704
minus 3 years = 20190704
day of the week = 1
is leap year = False
is weekday = True
the day, month and year = 4 7 2022
```

6.1.5 undated Functions

The `add_days`, `add_months` and `add_years` functions offer an alternative approach to the same functionality as the `YMD` class methods. With the YMD class being passed as a parameter.

```
import undated as ud
ymd = ud.YMD(2022_07_04)
print('ymd =', ymd)
print('plus 7 days =', ud.add_days(ymd, 7))
print('minus 7 days =', ud.add_days(ymd, -7))
print('add 2 months =', ud.add_months(ymd, 2))
print('minus 2 months =', ud.add_months(ymd, -2))
print('add 10 weekdays =', ud.add_weekdays(ymd, 10))
print('minus 10 weekdays =', ud.add_weekdays(ymd, -10))
```

gives the results

```
ymd = 20220704
plus 7 days = 20220711
minus 7 days = 20220627
add 2 months = 20220904
minus 2 months = 20220504
add 10 weekdays = 20220718
minus 10 weekdays = 20220620
```

The `undated` module also contains several *between* functions, that accept two YMD class objects. These calculate the days between two dates, the complete months between two dates or the weekdays, Monday to Friday, between two dates.

```
import undated as ud
ymd1 = ud.YMD(2022_07_04)
ymd2 = ud.YMD(2024_05_30)
print('ymd1 ymd2 =', ymd1, ymd2)
print('days between =', ud.days_between(ymd1, ymd2))
print('months between =', ud.months_between(ymd1, ymd2))
print('weekdays between =', ud.weekdays_between(ymd1, ymd2))
```

gives the results

```
ymd1 ymd2 = 20220704 20240530
days between = 696
months between = 22
weekdays between = 498
```

6.1.6 Format Deriver

The `udf.Deriver` class analyses the date data and tries to derive the format. It's designed with large amounts of data in mind, coming from various sources. It loops through the data until it finds a date that can be of only one format.

Note: The deriver has been designed to solve the problem where different data sources provide dates in different formats. The deriver assumes that all dates from the same data source, IE those passed to its search method, are all in the same format.

The following examples uses the tutorial.csv file on GitHub. Each date column contains dates in different formats, to represent the different data files being received.

In this example, the deriver is passed a column of date data, `date1` in this case, to derive.

```
import csv
import undated as ud
import undated.fmts as udf

with open('C:/Git/undated/docs/tutorial.csv', newline='') as csvfile:
    data = list(csv.DictReader(csvfile))
    dates = [row['date1'] for row in data]          # Get the required dates into a list
    deriver = udf.Deriver()                         # Initiate the deriver class
    deriver.set_parameters({udf.LANGUAGES: 'EN'})  # Set language to English, only␣
↪required for date3
    fmt = deriver.search(dates)                     # Search for the date format
    if fmt:
        for ymd in [ud.YMD(udf.as_parts(date, fmt)) for date in dates]:
            print(ymd)                              # The date is now an integer in Ymd␣
↪format
    else:
        print('Format not derived')
```

which gives the results

```
20200204
20210525
20220831
20080423
20060502
20200229
None
20211120
20201013
20210104
```

Changing the `date1` column to `date2` or `date3` will give the same output, as `udf.Deriver` will evaluate the correct date format.

Tip: Only pass enough dates to the search to be sure of getting a match. If there're thousands of rows, a few dozen may be enough to determine the format, and there's always the option of further searches.

6.1.7 Further Date Formats

The `Deriver` will try and derive the format from most common date presentations. The code below is definitely not how the package has been designed to be used but it does show the various date formats that can be accepted.

```
import undated as ud
import undated.fmts as udf


def go(lists_of_dates):
    for dates in lists_of_dates:
        fmt = udf.Deriver().search(dates)
        for sdate in dates:
            ymd_parts = udf.as_parts(sdate, fmt)
            print(ud.YMD(ymd_parts) if ymd_parts else 'Error', sdate, sep=' <- ')


go((
    ('20-mar-20', '21-apr-20', '22-may-20'),
    ('20mar20', '21apr20', '22may20'),
    ('11/25/2020 7:00PM Europe/Berlin',),
    ('25.11.2020 7:00PM Europe/Berlin',),
    ('Monday, 24 May 2021 05:50', 'Monday, 27 June 2021 05:50'),
    ('Mon, 25 Jan 2021 05:50:06 GMT', 'Mon, 27 Dec 2021 05:50:06 GMT'),
    ('Mon, 25 Jan 2021 05:50:06 GMT', 'Mon, 27 Dec 2021 05:50:06 GMT'),
    ('Mon, 25 Ene 2021 05:50:06 CET', 'Mon, 27 Dic 2021 05:50:06 CET'),
    ('12092022', '13092022'),
    ('2021-03-27T05:50:06.7199222-04:00',),
    ('03/28/2021 05:50:06',),
    ('29MAR2020', '01JAN2020'),
    ('Monday, 29 March 2021',),
    ('Monday, 29 March 2021 05:50 AM',),
    ('Monday, 29 March 2021 05:50:06',),
))
```

gives the results

```
20200320 <- 20-mar-20
20200421 <- 21-apr-20
20200522 <- 22-may-20
20200320 <- 20mar20
20200421 <- 21apr20
20200522 <- 22may20
20201125 <- 11/25/2020 7:00PM Europe/Berlin
20201125 <- 25.11.2020 7:00PM Europe/Berlin
20210529 <- Monday, 29 May 2021 05:50
20210629 <- Monday, 29 June 2021 05:50
20210129 <- Mon, 29 Jan 2021 05:50:06 GMT
20211229 <- Mon, 29 Dec 2021 05:50:06 GMT
20210129 <- Mon, 29 Jan 2021 05:50:06 GMT
20211229 <- Mon, 29 Dec 2021 05:50:06 GMT
20210129 <- Mon, 29 Ene 2021 05:50:06 CET
20211229 <- Mon, 29 Dic 2021 05:50:06 CET
20220912 <- 12092022
```

(continues on next page)

(continued from previous page)

```
20220913 <- 13092022
20210327 <- 2021-03-27T05:50:06.7199222-04:00
20210328 <- 03/28/2021 05:50:06
20200329 <- 29MAR2020
20200101 <- 01JAN2020
20210329 <- Monday, 29 March 2021
20210329 <- Monday, 29 March 2021 05:50 AM
20210329 <- Monday, 29 March 2021 05:50:06
```

6.1.8 Month Languages

The observant may have spotted some Spanish months in the last example. The `Deriver` currently caters for English, French, German and Spanish, full and abbreviated months names. If you know the language being used, setting it using the `set_parameters` method can improve performance. Which leads us on to…

6.1.9 Deriver set_parameters

To improve performance and assist with the format deriving process, the `Deriver` class object can have parameters set.

Hints

Hints help the `Deriver`, especially when there are fewer dates to use to derive the format. Current hints are:

- `udf.Y2` the year is two-digits
- `udf.YFIRST` the year is in the first position
- `udf.YLAST` the year is in the last position
- `udf.YM` the date only includes the year and month

The following code applies the hints for two-digit years, and the year in the last position.

```
import undated.fmts as udf
my_date = '200122'
deriver = udf.Deriver()
deriver.set_parameters({udf.HINTS: [udf.Y2, udf.YLAST]})
fmt = deriver.search(my_date)
print(udf.as_parts(my_date, fmt))
```

gives the result

```
(2022, 1, 20)
```

Languages

If dates have text based months, the language can be set if it is known. This will improve performance and accuracy.

```
import undated.fmts as udf
my_date = '20-JAN-2022'
deriver = udf.Deriver()
deriver.set_parameters({udf.LANGUAGES: 'EN1'})
fmt = deriver.search(my_date)
print(udf.as_parts(my_date, fmt))
```

gives the result

```
(2022, 1, 20)
```

In the above case, the format would not be derivable without specifying the language, as JAN could be English or German.

The language parameter above is `EN1`. The `EN` refers to the language, other valid options are `DE` German, `ES` Spanish and `FR` French. The `1` indicates that we are using the abbreviated months, `2` being for full month names. For example, `ES2` would be full Spanish month names, `FR1` would be abbreviated French months.

Time Separator

Often date strings include the time, which is out of scope for the `undated` package, so this needs to be removed. The default time separator character is `T`, following ISO standards. Space is also used. If dates have another separator character, this can be specified. In this example the `@` symbol has been used.

```
import undated.fmts as udf
my_date = '20-02-2022@12:55:55'
deriver = udf.Deriver()
deriver.set_parameters({udf.TIME_SEPARATOR: '@'})
fmt = deriver.search(my_date)
print(udf.as_parts(my_date, fmt))
```

gives the result

```
(2022, 2, 20)
```

YY Pivot

The `udf.YY_PIVOT` property is used to determine how the century is applied to two-digit years. By default, the `undated` pivot year is the current year minus 80. The default Excel pivot year is set as 40. The value should be a four-digit year. `1940` would mean any two-digit year 40 or over would be given the century `19`. Any two-digit year `39` and under will be given the century `20`.

```
import undated.fmts as udf
my_date = '20-01-35'
# Set the first deriver to 1940
deriver1 = udf.Deriver()
deriver1.set_parameters({udf.HINTS: [udf.Y2], udf.YY_PIVOT: 1940})
fmt1 = deriver1.search(my_date)
print(udf.as_parts(my_date, fmt1))
```

(continues on next page)

(continued from previous page)

```
# Now try again with the pivot at 1930
deriver2 = udf.Deriver()
deriver1.set_parameters({udf.HINTS: [udf.Y2], udf.YY_PIVOT: 1930})
fmt2 = deriver1.search(my_date)
print(udf.as_parts(my_date, fmt2))
```

gives the result

```
(2035, 1, 20)
(1935, 1, 20)
```

6.1.10 String Formats

The `UndatedFormat` object is not designed to be created manually. So, if the date format is simple and known, string-based formats can be used. These use the letters `YyMmd` along with the `-` character to indicate a separator.

- **Y**: 4-digit year
- **y**: 2-digit year
- **M**: the month as a string
- **m**: the month as an integer
- **d**: the day as an integer
- **-**: separator character, indicates a space, comma, dot, slash or dash

```
import undated.fmts as udf
print(udf.as_parts(2021_06_12, fmt='Ymd'))
print(udf.as_parts('2021JUN12', fmt='YMd'))
print(udf.as_parts('21JUN12', fmt='yMd'))
print(udf.as_parts('12/JUN/2021', fmt='d-M-Y'))
print(udf.as_parts('12-JUN-21', fmt='d-M-y'))
print(udf.as_parts('12.06.21', fmt='d-m-y'))
```

gives the result (for each print)

```
(2021, 6, 12)
```

To go the next step and convert to `ud.YMD` or `datetime`

```
import datetime
import undated as ud
import undated.fmts as udf

parts = udf.as_parts('12-JUN-21', fmt='d-M-y')
print(ud.YMD(parts))
print(datetime.datetime(*parts))
```

gives the result

```
20210612
2021-06-12 00:00:00
```

6.2 undated

The main module for manipulating dates that are not `datetime` datatypes. Consisting of a main class object, `YMD`, for managing a date as an integer and several functions for further functionality, such as adding or calculating differences. All of these are hopefully self explanitory.

class undated.**YMD**(*year_or_iymd: Optional[Union[int, tuple]]*, *month: Optional[int] = None*, *day: Optional[int] = None*, *trusted: bool = False*)

Class of date parts, year, month, day

Parameters

- **year_or_iymd** – the year, date in Ymd format or tuple of date parts
- **month** – the month number, 1 = Jan, defaults to 1
- **day** – the day. If not supplied, the 1st is assumed
- **trusted** – True, if dates can be trusted to be correct, the validation stage is skipped

add_days(*days: int*) → *YMD*

Adds the specified number of days to the date. Another option would be to use the addition operator `ymd2 = ymd1 + 2`

Parameters
days – The number of days to add. Use negative days to subtract days.

Returns
YMD class object

add_months(*months: int*, *period: bool = False*) → *YMD*

Adds the specified number of months to the date

Parameters

- **months** – The number of months to add, use negative months to subtract
- **period** – Set to True when looking for a period end date. So… With period False, Jan 1st + 12 months, would be 1st Jan the following year. With period set to True, it would be 31st Dec the same year.

Returns
YMD class object

add_weekdays(*weekdays: int*) → *YMD*

Adds the specified number of weekdays, Monday to Friday, to the date

Parameters
weekdays – The number of weekdays to add. Use negative days to subtract.

Returns
YMD class object

add_years(*years: int*, *period: bool = False*) → *YMD*

Adds a specified number of years to the date

Parameters

- **years** – The number of years to add. Use negative years to subtract
- **period** – Set to True when looking for a period end date. So… With period False, Jan 1st + 1 year, would be 1st Jan the following year. With period set to True, it would be 31st Dec the same year.

Returns
YMD class object

day: Optional[int]
The day element of the date, as a 1 or 2 digit integer

day_of_week() → Optional[int]
The number for the day of the week. Sunday == 0, Monday == 1…

Returns
The day number 0 to 6

is_leap_year() → bool
Is the date falling within a leap year

Returns
True when the date falls in a leap year

is_weekday() → bool
Is the date falling on a weekday, IE between Monday and Friday

Returns
True when it is a weekday

iymd: Optional[int]
The date as an 8 digit integer, in year, month, day format

month: Optional[int]
The month element of the date, as a 1 or 2 digit integer. January==1, December==12

status: int = None
The status of the class. Refers to the package constants VALID, INVALID and TRUSTED

year: Optional[int]
The year element of the date, as a 4 digit integer

undated.**add_days**(*ymd:* YMD, *days: int*) → *YMD*
Adds a number of days to a date in in Ymd format

Parameters

- **ymd** – YMD class object
- **days** – int, the number of days to add

Returns
YMD class object

undated.**add_months**(*ymd:* YMD, *months: int*, *period: bool = False*) → *YMD*
Adds given months to a date. Use negative months to subtract months

Parameters

- **ymd** – YMD class object
- **months** – int, the number of months
- **period** – bool, takes the previous day. EG: For last day of a period

Returns
YMD class object

undated.**add_weekdays**(*ymd: YMD*, *weekdays: int*) → *YMD*

Adds a number of weekdays, monday to friday, to a date in in Ymd format

Parameters

- **ymd** – YMD class object
- **weekdays** – int, the number of days to add

Returns
YMD class object

undated.**days_between**(*from_ymd: YMD*, *to_ymd: YMD*) → Optional[int]

Calculates the days between two dates

Parameters

- **from_ymd** – YMD, the from date YMD class object
- **to_ymd** – YMD, the to date YMD class object

Returns
int, the days between the dates

undated.**months_between**(*from_ymd: YMD*, *to_ymd: YMD*) → Optional[int]

Calculates the complete months between two dates

Parameters

- **from_ymd** – YMD class object
- **to_ymd** – YMD class object

Returns
int, the complete months between the dates

undated.**quarter**(*ymd: YMD*, *to_str: bool = True*) → Optional[Union[int, str]]

Calculates the quarter from a year, returning the quarter end month, or quarter number

Parameters

- **ymd** – YMD class object
- **to_str** – bool, true returns 2021Q3, otherwise 202103 format

Returns
str 2021Q1, 2021Q2, 2021Q3, 2021Q4; or int 202103, 202106, 202109, 202112

undated.**weekdays_between**(*from_ymd: YMD*, *to_ymd: YMD*, *inclusive: bool = False*) → Optional[int]

Calculates the number of weekdays between two dates

Parameters

- **from_ymd** – YMD class object
- **to_ymd** – YMD class object
- **inclusive** – bool, whether to include the to date as a completed day

Returns
int, the number of days between the dates

6.3 undated.fmts

The `fmts` (formattings) module comes into play when the date data format is unknown. It has two key class objects. The `Deriver` class derives the date format from a list of dates, returning an `UndatedFormat` class object, which contains the information required by the `as_parts` function to extract the date elements.

class undated.fmts.**Deriver**

Facilitates the searching of dates to derive the format

search(*dates: Union[list, str, tuple]*) → Optional[*UndatedFormat*]

Search through a list of dates to derive the date format

> **Caution:** All of the dates in the list passed to the search method are expected to be in the same format.

Parameters
dates – list or tuple of dates to search. Or str for one date

Returns
the derived `UndatedFormat` object

set_parameters(*params: dict*)

Sets the optional parameters for the search

Parameters
params – see tutorial for possible parameters

class undated.fmts.**UndatedFormat**(*split: list[int, int, int]*, *keys: list[str, str, str]*, *steps: dict*, *valid: bool*)

Properties for the format. Created by the `Deriver` class or `convert_format` function

undated.fmts.**as_parts**(*sdate: Union[int, str]*, *fmt: Union[str,* UndatedFormat*]*, *yy_pivot: Optional[int] = None*) → Optional[tuple]

Converts the sdate to year, month, day, based on the format

Parameters

- **sdate** – The date as a str or int
- **fmt** – The date format, as either a basic format as a string, or a derived format
- **yy_pivot** – The pivot year for two digit years. Use with string based formats

undated.fmts.**convert_format**(*fmt: str*, *yy_pivot: Optional[int] = None*) → *UndatedFormat*

Converts the basic string format into an `UndatedFormat` object. Recommended when looping, to prevent repeated format conversion.

Parameters

- **fmt** – The string format. See the tutorial for valid values.
- **yy_pivot** – The pivot year for two digit years. Use only with string based formats.

6.4 undated.utils

The utils module functionality mirrors that of the `ud.YMD` class. However these functions have been stripped down to improve performance. Use only when dates are valid integers in the `Ymd` format.

`undated.utils.`**`add_days`**(*iymd: int*, *days: int*) → int

Adds a number of days to a date in in Ymd format

Parameters

- **`iymd`** – the date in Ymd format
- **`days`** – the number of days to add

Returns
the new date in Ymd format

`undated.utils.`**`add_months`**(*iymd: int*, *months: int*) → int

Adds given months to a date. Use negative months to subtract months

Parameters

- **`iymd`** – the date in Ymd format
- **`months`** – the number of months

Returns
the new date

`undated.utils.`**`add_weekdays`**(*iymd: int*, *weekdays: int*) → int

Adds a number of weekdays, monday to friday, to a date in in Ymd format

Parameters

- **`iymd`** – the date in Ymd format
- **`weekdays`** – the number of days to add

Returns
the new date in Ymd format

`undated.utils.`**`day_of_week`**(*iymd: int*) → int

Calculates the number for day of the week. Sunday = 0, Monday = 1…

Parameters
`iymd` – date in Ymd format

Returns
the day number 0 to 6

`undated.utils.`**`days_between`**(*from_iymd: int*, *to_iymd: int*) → int

Calculates the days between two dates

Parameters

- **`from_iymd`** – the from date in Ymd format
- **`to_iymd`** – the to date in Ymd format

Returns
the days between the dates

undated.utils.**first_day**(*iym: int*) → int

Converts a year month integer to a year month day integer, as at the first day of the month Simple formula, exists for completion.

Parameters
iym – The year month in Ym format

Returns
The year month day in Ymd format

undated.utils.**is_leap_year**(*year: int*) → int

Is the year a leap year

Parameters
year – the year

Returns
1 for leap year, 0 not a leap year

undated.utils.**is_valid**(*iymd: int*) → bool

Checks the date is valid. Year expected to be between 1583 and 9999

Parameters
iymd – the date in Ymd format

Returns
is the date is a valid date or not

undated.utils.**is_weekday**(*iymd: int*) → bool

Calculates if the date is a weekday, Monday - Friday

Parameters
iymd – the date in Ymd format

Returns
True when it is a weekday

undated.utils.**last_day**(*iym: int*) → int

Converts a year month integer to a year month day integer, as at the last day of the month

Parameters
iym – The year month in Ym format

Returns
The year month day in Ymd format

undated.utils.**months_between**(*from_iymd: Union[int,* YMD*]*, *to_iymd: Union[int,* YMD*]*) → int

Calculates the complete months between two dates

Parameters

- **from_iymd** – the from date in Ymd or Ym format
- **to_iymd** – the from date in Ymd or Ym format

Returns
the complete months between the dates

undated.utils.**quarter**(*iymd: int*, *to_str: bool = True*) → Union[int, str]

Calculates the quarter from a year, returning the quarter end month, or quarter number

Parameters

- **iymd** – the date in Ymd or Ym format
- **to_str** – true returns 2021Q3, otherwise 202103 format

Returns
str 2021Q1, 2021Q2, 2021Q3, 2021Q4; or int 202103, 202106, 202109, 202112

undated.utils.**weekdays_between**(*from_iymd: int*, *to_iymd: int*, *inclusive: bool = False*) → int

Calculates the complete months between two dates

Parameters

- **from_iymd** – the from date in Ymd format
- **to_iymd** – the from date in Ymd format
- **inclusive** – whether to include the to date as a completed day

Returns
the complete months between the dates

6.5 Change Log

6.5.1 Version 1.0.8

Date 10th October 2022

Added new functions to utils, *first_day* and *last_day*. These convert year month integers to year month day integers, with the day element being either the first or last day of the month.

6.6 Index

PYTHON MODULE INDEX

U

INDEX