When.py Documentation

Release 0.3.0

Andy Dirnberger

January 10, 2016

Contents

When.py provides user-friendly functions to help perform common date and time actions.

CHAPTER 1

Usage

Friendly Dates and Times

when.**all_timezones**()
Get a list of all time zones.

This is a wrapper for pytz.all_timezones.

Returns list – all time zones.

New in version 0.1.0.

when.**all_timezones_set**()
Get a set of all time zones.

This is a wrapper for pytz.all_timezones_set.

Returns set – all time zones.

New in version 0.1.0.

when.**common_timezones**()
Get a list of common time zones.

This is a wrapper for pytz.common_timezones.

Returns list – common time zones.

New in version 0.1.0.

when.**common_timezones_set**()
Get a set of common time zones.

This is a wrapper for pytz.common_timezones_set.

Returns set – common time zones.

New in version 0.1.0.

when.**ever**()
Get a random datetime.

Instead of using datetime.MINYEAR and datetime.MAXYEAR as the bounds, the current year +/- 100 is used. The thought behind this is that years that are too extreme will not be as useful.

Returns datetime.datetime – a random datetime.

New in version 0.3.0.

when.**format**(*value*, *format_string*)
Get a formatted version of a datetime.

This is a wrapper for `strftime()`. The full list of directives that can be used can be found at http://docs.python.org/library/datetime.html#strftime-strptime-behavior. Predefined formats are exposed through `when.formats`:

when.formats.**DATE**
Date in locale-based format.

when.formats.**DATETIME**
Date and time in locale-based format.

when.formats.**TIME**
Time in locale-based format.

when.formats.**TIME_AMPM**
12-hour time in locale-based format.

Parameters

- **value** (*datetime.datetime, datetime.date, datetime.time.*) – A datetime object.
- **format_string** (*str.*) – A string specifying formatting the directives or to use.

Returns str – the formatted datetime.

Raises AssertionError

New in version 0.3.0.

when.**future**(*years=0*, *months=0*, *weeks=0*, *days=0*, *hours=0*, *minutes=0*, *seconds=0*, *milliseconds=0*, *microseconds=0*, *utc=False*)
Get a datetime in the future.

`future()` accepts the all of the parameters of `datetime.timedelta`, plus includes the parameters `years` and `months`. `years` and `months` will add their respective units of time to the datetime.

By default `future()` will return the datetime in the system's local time. If the `utc` parameter is set to `True` or `set_utc()` has been called, the datetime will be based on UTC instead.

Parameters

- **years** (*int.*) – The number of years to add.
- **months** (*int.*) – The number of months to add.
- **weeks** (*int.*) – The number of weeks to add.
- **days** (*int.*) – The number of days to add.
- **hours** (*int.*) – The number of hours to add.
- **minutes** (*int.*) – The number of minutes to add.
- **seconds** (*int.*) – The number of seconds to add.
- **milliseconds** (*int.*) – The number of milliseconds to add.
- **microseconds** (*int.*) – The number of microseconds to add.
- **utc** (*bool.*) – Whether or not to use UTC instead of local time.

Returns datetime.datetime – the calculated datetime.

New in version 0.1.0.

when.**how_many_leap_days**(*from_date*, *to_date*)
Get the number of leap days between two dates

Parameters

- **from_date** (*datetime.datetime, datetime.date*) – A datetime object. If only a year is specified, will use January 1.
- **to_date** (*datetime.datetime, datetime.date*) – A datetime object.. If only a year is specified, will use January 1.

Returns int – the number of leap days.

New in version 0.3.0.

when.**is_timezone_aware**(*value*)
Check if a datetime is time zone aware.

is_timezone_aware() is the inverse of *is_timezone_naive()*.

Parameters **value** (*datetime.datetime, datetime.time*) – A valid datetime object.

Returns bool – if the object is time zone aware.

New in version 0.3.0.

when.**is_timezone_naive**(*value*)
Check if a datetime is time zone naive.

is_timezone_naive() is the inverse of *is_timezone_aware()*.

Parameters **value** (*datetime.datetime, datetime.time*) – A valid datetime object.

Returns bool – if the object is time zone naive.

New in version 0.3.0.

when.**now**(*utc=False*)
Get a datetime representing the current date and time.

By default `now()` will return the datetime in the system's local time. If the `utc` parameter is set to `True` or `set_utc()` has been called, the datetime will be based on UTC instead.

Parameters **utc** (*bool.*) – Whether or not to use UTC instead of local time.

Returns datetime.datetime – the current datetime.

New in version 0.1.0.

when.**past**(*years=0*, *months=0*, *weeks=0*, *days=0*, *hours=0*, *minutes=0*, *seconds=0*, *milliseconds=0*, *microseconds=0*, *utc=False*)
Get a datetime in the past.

`past()` accepts the all of the parameters of `datetime.timedelta`, plus includes the parameters `years` and `months`. `years` and `months` will add their respective units of time to the datetime.

By default `past()` will return the datetime in the system's local time. If the `utc` parameter is set to `True` or `set_utc()` has been called, the datetime will be based on UTC instead.

Parameters

- **years** (*int.*) – The number of years to subtract.
- **months** (*int.*) – The number of months to subtract.
- **weeks** (*int.*) – The number of weeks to subtract.
- **days** (*int.*) – The number of days to subtract.

- **hours** (*int.*) – The number of hours to subtract.
- **minutes** (*int.*) – The number of minutes to subtract.
- **seconds** (*int.*) – The number of seconds to subtract.
- **milliseconds** (*int.*) – The number of milliseconds to subtract.
- **microseconds** (*int.*) – The number of microseconds to subtract.
- **utc** (*bool.*) – Whether or not to use UTC instead of local time.

Returns datetime.datetime – the calculated datetime.

New in version 0.1.0.

when.**set_utc**()
Set all datetimes to UTC.

The `utc` parameter of other methods will be ignored, with the global setting taking precedence.

This can be reset by calling `unset_utc()`.

New in version 0.1.0.

when.**shift**(*value*, *from_tz=None*, *to_tz=None*, *utc=False*)
Convert a datetime from one time zone to another.

`value` will be converted from its time zone (when it is time zone aware) or the time zone specified by `from_tz` (when it is time zone naive) to the time zone specified by `to_tz`. These values can either be strings containing the name of the time zone (see `pytz.all_timezones` for a list of all supported values) or a `datetime.tzinfo` object.

If no value is provided for either `from_tz` (when `value` is time zone naive) or `to_tz`, the current system time zone will be used. If the `utc` parameter is set to `True` or `set_utc()` has been called, however, UTC will be used instead.

Parameters

- **value** (*datetime.datetime, datetime.time.*) – A datetime object.
- **from_tz** (*datetime.tzinfo, str.*) – The time zone to shift from.
- **to_tz** (*datetime.tzinfo, str.*) – The time zone to shift to.
- **utc** (*bool.*) – Whether or not to use UTC instead of local time.

Returns datetime.datetime – the calculated datetime.

Raises AssertionError

Changed in version 0.3.0: Added AssertionError for invalid values of `value`

when.**timezone**()
Get the name of the current system time zone.

Returns str – the name of the system time zone.

New in version 0.1.0.

when.**timezone_object**(*tz_name=None*)
Get the current system time zone.

Parameters **tz_name** (*str.*) – The name of the time zone.

Returns datetime.tzinfo – the time zone, defaults to system time zone.

New in version 0.1.0.

when.**today**()
Get a date representing the current date.

Returns datetime.date – the current date.

New in version 0.1.0.

when.**tomorrow**()
Get a date representing tomorrow's date.

Returns datetime.date – the current date plus one day.

New in version 0.1.0.

when.**unset_utc**()
Set all datetimes to system time.

The `utc` parameter of other methods will be used.

This can be changed by calling `set_utc()`.

New in version 0.1.0.

when.**yesterday**()
Get a date representing yesterday's date.

Returns datetime.date – the current date minus one day.

New in version 0.1.0.

CHAPTER 2

A note about `future` and `past`

When changing a datetime from one month (or year) to another, it is often the case that the new month will have fewer days than the original, resulting in an invalid date. When this happens, the days will be adjusted into the future. This is consistent with implementations found elsewhere.

```
>>> when.today()
datetime.date(2012, 2, 29)
>>>
>>> when.future(years=1)
datetime.datetime(2013, 3, 1, 19, 0, 23, 76878)
```

```
>>> when.today()
datetime.date(2012, 3, 31)
>>>
>>> when.past(months=1)
datetime.datetime(2012, 3, 2, 19, 7, 36, 317653)
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

W