When *data* is an Index or Series, the underlying array will be extracted from *data*.

dtype [str, np.dtype, or ExtensionDtype, optional] The dtype to use for the array. This may be a NumPy dtype or an extension type registered with pandas using *pandas.api.extensions.register_extension_dtype()*.

If not specified, there are two possibilities:

1. When *data* is a *Series*, *Index*, or `ExtensionArray`, the *dtype* will be taken from the data.
2. Otherwise, pandas will attempt to infer the *dtype* from the data.

Note that when *data* is a NumPy array, `data.dtype` is *not* used for inferring the array type. This is because NumPy cannot represent all the types of data that can be held in extension arrays.

Currently, pandas will infer an extension dtype for sequences of

Scalar Type	Array Type
pandas.Interval	*pandas.arrays.IntervalArray*
pandas.Period	*pandas.arrays.PeriodArray*
datetime.datetime	*pandas.arrays.DatetimeArray*
datetime.timedelta	*pandas.arrays.TimedeltaArray*
int	*pandas.arrays.IntegerArray*
str	*pandas.arrays.StringArray*
bool	*pandas.arrays.BooleanArray*

For all other cases, NumPy's usual inference rules will be used.

Changed in version 1.0.0: Pandas infers nullable-integer dtype for integer data, string dtype for string data, and nullable-boolean dtype for boolean data.

copy [bool, default True] Whether to copy the data, even if not necessary. Depending on the type of *data*, creating the new array may require copying data, even if `copy=False`.

Returns

ExtensionArray The newly created array.

Raises

ValueError When *data* is not 1-dimensional.

See also:

numpy.array Construct a NumPy array.
Series Construct a pandas Series.
Index Construct a pandas Index.
arrays.PandasArray ExtensionArray wrapping a NumPy array.
Series.array Extract the array stored within a Series.

Notes

Omitting the *dtype* argument means pandas will attempt to infer the best array type from the values in the data. As new array types are added by pandas and 3rd party libraries, the "best" array type may change. We recommend specifying *dtype* to ensure that

1. the correct array type for the data is returned
2. the returned array type doesn't change as new extension types are added by pandas and third-party libraries

Additionally, if the underlying memory representation of the returned array matters, we recommend specifying the *dtype* as a concrete object rather than a string alias or allowing it to be inferred. For example, a future version of pandas or a 3rd-party library may include a dedicated ExtensionArray for string data. In this event, the following would no longer return a *arrays.PandasArray* backed by a NumPy array.

```
>>> pd.array(['a', 'b'], dtype=str)
<PandasArray>
['a', 'b']
Length: 2, dtype: str32
```

This would instead return the new ExtensionArray dedicated for string data. If you really need the new array to be backed by a NumPy array, specify that in the dtype.

```
>>> pd.array(['a', 'b'], dtype=np.dtype("<U1"))
<PandasArray>
['a', 'b']
Length: 2, dtype: str32
```

Finally, Pandas has arrays that mostly overlap with NumPy

- *arrays.DatetimeArray*
- *arrays.TimedeltaArray*

When data with a `datetime64[ns]` or `timedelta64[ns]` dtype is passed, pandas will always return a `DatetimeArray` or `TimedeltaArray` rather than a `PandasArray`. This is for symmetry with the case of timezone-aware data, which NumPy does not natively support.

```
>>> pd.array(['2015', '2016'], dtype='datetime64[ns]')
<DatetimeArray>
['2015-01-01 00:00:00', '2016-01-01 00:00:00']
Length: 2, dtype: datetime64[ns]
```

```
>>> pd.array(["1H", "2H"], dtype='timedelta64[ns]')
<TimedeltaArray>
['01:00:00', '02:00:00']
Length: 2, dtype: timedelta64[ns]
```

Examples

If a dtype is not specified, pandas will infer the best dtype from the values. See the description of *dtype* for the types pandas infers for.

```
>>> pd.array([1, 2])
<IntegerArray>
[1, 2]
Length: 2, dtype: Int64
```

```
>>> pd.array([1, 2, np.nan])
<IntegerArray>
```

(continues on next page)

(continued from previous page)

```
[1, 2, NaN]
Length: 3, dtype: Int64
```

```
>>> pd.array(["a", None, "c"])
<StringArray>
['a', nan, 'c']
Length: 3, dtype: string
```

```
>>> pd.array([pd.Period('2000', freq="D"), pd.Period("2000", freq="D")])
<PeriodArray>
['2000-01-01', '2000-01-01']
Length: 2, dtype: period[D]
```

You can use the string alias for *dtype*

```
>>> pd.array(['a', 'b', 'a'], dtype='category')
[a, b, a]
Categories (2, object): [a, b]
```

Or specify the actual dtype

```
>>> pd.array(['a', 'b', 'a'],
...          dtype=pd.CategoricalDtype(['a', 'b', 'c'], ordered=True))
[a, b, a]
Categories (3, object): [a < b < c]
```

If pandas does not infer a dedicated extension type a *arrays.PandasArray* is returned.

```
>>> pd.array([1.1, 2.2])
<PandasArray>
[1.1, 2.2]
Length: 2, dtype: float64
```

As mentioned in the "Notes" section, new extension types may be added in the future (by pandas or 3rd party libraries), causing the return value to no longer be a *arrays.PandasArray*. Specify the *dtype* as a NumPy dtype if you need to ensure there's no future change in behavior.

```
>>> pd.array([1, 2], dtype=np.dtype("int32"))
<PandasArray>
[1, 2]
Length: 2, dtype: int32
```

data must be 1-dimensional. A ValueError is raised when the input has the wrong dimensionality.

```
>>> pd.array(1)
Traceback (most recent call last):
  ...
ValueError: Cannot pass scalar '1' to 'pandas.array'.
```

3.5.2 Datetime data

NumPy cannot natively represent timezone-aware datetimes. Pandas supports this with the *arrays.DatetimeArray* extension array, which can hold timezone-naive or timezone-aware values.

Timestamp, a subclass of `datetime.datetime`, is pandas' scalar type for timezone-naive or timezone-aware datetime data.

Timestamp([ts_input, freq, tz, unit, year, ...])	Pandas replacement for python datetime.datetime object.

pandas.Timestamp

class pandas.**Timestamp**(*ts_input=<object object>*, *freq=None*, *tz=None*, *unit=None*, *year=None*, *month=None*, *day=None*, *hour=None*, *minute=None*, *second=None*, *microsecond=None*, *nanosecond=None*, *tzinfo=None*)

Pandas replacement for python datetime.datetime object.

Timestamp is the pandas equivalent of python's Datetime and is interchangeable with it in most cases. It's the type used for the entries that make up a DatetimeIndex, and other timeseries oriented data structures in pandas.

Parameters

ts_input [datetime-like, str, int, float] Value to be converted to Timestamp.

freq [str, DateOffset] Offset which Timestamp will have.

tz [str, pytz.timezone, dateutil.tz.tzfile or None] Time zone for time which Timestamp will have.

unit [str] Unit used for conversion if ts_input is of type int or float. The valid values are 'D', 'h', 'm', 's', 'ms', 'us', and 'ns'. For example, 's' means seconds and 'ms' means milliseconds.

year, month, day [int]

hour, minute, second, microsecond [int, optional, default 0]

nanosecond [int, optional, default 0] New in version 0.23.0.

tzinfo [datetime.tzinfo, optional, default None]

Notes

There are essentially three calling conventions for the constructor. The primary form accepts four parameters. They can be passed by position or keyword.

The other two forms mimic the parameters from `datetime.datetime`. They can be passed by either position or keyword, but not both mixed together.

Examples

Using the primary calling convention:

This converts a datetime-like string

```
>>> pd.Timestamp('2017-01-01T12')
Timestamp('2017-01-01 12:00:00')
```

This converts a float representing a Unix epoch in units of seconds

```
>>> pd.Timestamp(1513393355.5, unit='s')
Timestamp('2017-12-16 03:02:35.500000')
```

This converts an int representing a Unix-epoch in units of seconds and for a particular timezone

```
>>> pd.Timestamp(1513393355, unit='s', tz='US/Pacific')
Timestamp('2017-12-15 19:02:35-0800', tz='US/Pacific')
```

Using the other two forms that mimic the API for `datetime.datetime`:

```
>>> pd.Timestamp(2017, 1, 1, 12)
Timestamp('2017-01-01 12:00:00')
```

```
>>> pd.Timestamp(year=2017, month=1, day=1, hour=12)
Timestamp('2017-01-01 12:00:00')
```

Attributes

asm8	Return numpy datetime64 format in nanoseconds.
dayofweek	Return day of the week.
dayofyear	Return the day of the year.
days_in_month	Return the number of days in the month.
daysinmonth	Return the number of days in the month.
freqstr	Return the total number of days in the month.
is_leap_year	Return True if year is a leap year.
is_month_end	Return True if date is last day of month.
is_month_start	Return True if date is first day of month.
is_quarter_end	Return True if date is last day of the quarter.
is_quarter_start	Return True if date is first day of the quarter.
is_year_end	Return True if date is last day of the year.
is_year_start	Return True if date is first day of the year.
quarter	Return the quarter of the year.
tz	Alias for tzinfo.
week	Return the week number of the year.
weekofyear	Return the week number of the year.

pandas.Timestamp.asm8

Timestamp.**asm8**
Return numpy datetime64 format in nanoseconds.

pandas.Timestamp.dayofweek

property Timestamp.**dayofweek**
Return day of the week.

pandas.Timestamp.dayofyear

property Timestamp.**dayofyear**
Return the day of the year.

pandas.Timestamp.days_in_month

property Timestamp.**days_in_month**
Return the number of days in the month.

pandas.Timestamp.daysinmonth

property Timestamp.**daysinmonth**
Return the number of days in the month.

pandas.Timestamp.freqstr

property Timestamp.**freqstr**
Return the total number of days in the month.

pandas.Timestamp.is_leap_year

property Timestamp.**is_leap_year**
Return True if year is a leap year.

pandas.Timestamp.is_month_end

property Timestamp.**is_month_end**
Return True if date is last day of month.

pandas.Timestamp.is_month_start

property Timestamp.**is_month_start**
Return True if date is first day of month.

pandas.Timestamp.is_quarter_end

property Timestamp.**is_quarter_end**
Return True if date is last day of the quarter.

pandas.Timestamp.is_quarter_start

property Timestamp.**is_quarter_start**
Return True if date is first day of the quarter.

pandas.Timestamp.is_year_end

property Timestamp.**is_year_end**
Return True if date is last day of the year.

pandas.Timestamp.is_year_start

property Timestamp.**is_year_start**
Return True if date is first day of the year.

pandas.Timestamp.quarter

property Timestamp.**quarter**
Return the quarter of the year.

pandas.Timestamp.tz

property Timestamp.**tz**
Alias for tzinfo.

pandas.Timestamp.week

property Timestamp.**week**
Return the week number of the year.

pandas.Timestamp.weekofyear

property Timestamp.**weekofyear**
Return the week number of the year.

day	
fold	
freq	
hour	
microsecond	
minute	
month	
nanosecond	
second	
tzinfo	
value	
year	

Methods

astimezone(self, tz)	Convert tz-aware Timestamp to another time zone.
ceil(self, freq[, ambiguous, nonexistent])	return a new Timestamp ceiled to this resolution.
combine(date, time)	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_name(self[, locale])	Return the day name of the Timestamp with specified locale.
dst()	Return self.tzinfo.dst(self).
floor(self, freq[, ambiguous, nonexistent])	return a new Timestamp floored to this resolution.
fromisoformat()	string -> datetime from datetime.isoformat() output
fromordinal(ordinal[, freq, tz])	Passed an ordinal, translate and convert to a ts.
fromtimestamp(ts)	timestamp[, tz] -> tz's local time from POSIX timestamp.
isocalendar()	Return a 3-tuple containing ISO year, week number, and weekday.
isoweekday()	Return the day of the week represented by the date.
month_name(self[, locale])	Return the month name of the Timestamp with specified locale.
normalize(self)	Normalize Timestamp to midnight, preserving tz information.
now([tz])	Return new Timestamp object representing current time local to tz.
replace(self[, year, month, day, hour, ...])	implements datetime.replace, handles nanoseconds.
round(self, freq[, ambiguous, nonexistent])	Round the Timestamp to the specified resolution.
strftime()	format -> strftime() style string.
strptime(string, format)	Function is not implemented.
time()	Return time object with same time but with tzinfo=None.
timestamp()	Return POSIX timestamp as float.

continues on next page

Table 82 – continued from previous page

timetuple()	Return time tuple, compatible with time.localtime().
timetz()	Return time object with same time and tzinfo.
to_datetime64()	Return a numpy.datetime64 object with 'ns' precision.
to_julian_date(self)	Convert TimeStamp to a Julian Date.
to_numpy()	Convert the Timestamp to a NumPy datetime64.
to_period(self[, freq])	Return an period of which this timestamp is an observation.
to_pydatetime()	Convert a Timestamp object to a native Python datetime object.
today(cls[, tz])	Return the current time in the local timezone.
toordinal()	Return proleptic Gregorian ordinal.
tz_convert(self, tz)	Convert tz-aware Timestamp to another time zone.
tz_localize(self, tz[, ambiguous, nonexistent])	Convert naive Timestamp to local time zone, or remove timezone from tz-aware Timestamp.
tzname()	Return self.tzinfo.tzname(self).
utcfromtimestamp(ts)	Construct a naive UTC datetime from a POSIX timestamp.
utcnow()	Return a new Timestamp 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.

pandas.Timestamp.astimezone

Timestamp.**astimezone**(*self*, *tz*)

Convert tz-aware Timestamp to another time zone.

Parameters

tz [str, pytz.timezone, dateutil.tz.tzfile or None] Time zone for time which Timestamp will be converted to. None will remove timezone holding UTC time.

Returns

converted [Timestamp]

Raises

TypeError If Timestamp is tz-naive.

pandas.Timestamp.ceil

Timestamp.**ceil**(*self*, *freq*, *ambiguous='raise'*, *nonexistent='raise'*)

return a new Timestamp ceiled to this resolution.

Parameters

freq [str] Frequency string indicating the ceiling resolution.

ambiguous [bool or {'raise', 'NaT'}, default 'raise'] The behavior is as follows:

- bool contains flags to determine if time is dst or not (note that this flag is only applicable for ambiguous fall dst dates).
- 'NaT' will return NaT for an ambiguous time.
- 'raise' will raise an AmbiguousTimeError for an ambiguous time.

New in version 0.24.0.

nonexistent [{'raise', 'shift_forward', 'shift_backward, 'NaT', timedelta}, default 'raise'] A nonexistent time does not exist in a particular timezone where clocks moved forward due to DST.

- 'shift_forward' will shift the nonexistent time forward to the closest existing time.
- 'shift_backward' will shift the nonexistent time backward to the closest existing time.
- 'NaT' will return NaT where there are nonexistent times.
- timedelta objects will shift nonexistent times by the timedelta.
- 'raise' will raise an NonExistentTimeError if there are nonexistent times.

New in version 0.24.0.

Raises

ValueError if the freq cannot be converted.

pandas.Timestamp.combine

classmethod Timestamp.**combine**(*date*, *time*)
date, time -> datetime with same date and time fields.

pandas.Timestamp.ctime

Timestamp.**ctime**()
Return ctime() style string.

pandas.Timestamp.date

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

pandas.Timestamp.day_name

Timestamp.**day_name**(*self*, *locale=None*) → 'unicode'
Return the day name of the Timestamp with specified locale.

Parameters

locale [string, default None (English locale)] Locale determining the language in which to return the day name.

Returns

day_name [string]

New in version 0.23.0: ..

pandas.Timestamp.dst

`Timestamp.dst()`
Return self.tzinfo.dst(self).

pandas.Timestamp.floor

`Timestamp.floor(self, freq, ambiguous='raise', nonexistent='raise')`
return a new Timestamp floored to this resolution.

Parameters

freq [str] Frequency string indicating the flooring resolution.

ambiguous [bool or {'raise', 'NaT'}, default 'raise'] The behavior is as follows:

- bool contains flags to determine if time is dst or not (note that this flag is only applicable for ambiguous fall dst dates).
- 'NaT' will return NaT for an ambiguous time.
- 'raise' will raise an AmbiguousTimeError for an ambiguous time.

New in version 0.24.0.

nonexistent [{'raise', 'shift_forward', 'shift_backward, 'NaT', timedelta}, default 'raise'] A nonexistent time does not exist in a particular timezone where clocks moved forward due to DST.

- 'shift_forward' will shift the nonexistent time forward to the closest existing time.
- 'shift_backward' will shift the nonexistent time backward to the closest existing time.
- 'NaT' will return NaT where there are nonexistent times.
- timedelta objects will shift nonexistent times by the timedelta.
- 'raise' will raise an NonExistentTimeError if there are nonexistent times.

New in version 0.24.0.

Raises

ValueError if the freq cannot be converted.

pandas.Timestamp.fromisoformat

Timestamp.**fromisoformat**()
string -> datetime from datetime.isoformat() output

pandas.Timestamp.fromordinal

classmethod Timestamp.**fromordinal**(*ordinal*, *freq=None*, *tz=None*)
Passed an ordinal, translate and convert to a ts. Note: by definition there cannot be any tz info on the ordinal itself.

Parameters

ordinal [int] Date corresponding to a proleptic Gregorian ordinal.

freq [str, DateOffset] Offset to apply to the Timestamp.

tz [str, pytz.timezone, dateutil.tz.tzfile or None] Time zone for the Timestamp.

pandas.Timestamp.fromtimestamp

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

pandas.Timestamp.isocalendar

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

pandas.Timestamp.isoweekday

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

pandas.Timestamp.month_name

Timestamp.**month_name**(*self*, *locale=None*) → 'unicode'
Return the month name of the Timestamp with specified locale.

Parameters

locale [string, default None (English locale)] Locale determining the language in which to return the month name.

Returns

month_name [string]

New in version 0.23.0: ..

pandas.Timestamp.normalize

Timestamp.**normalize**(*self*)

Normalize Timestamp to midnight, preserving tz information.

pandas.Timestamp.now

classmethod Timestamp.**now**(*tz=None*)

Return new Timestamp object representing current time local to tz.

Parameters

tz [str or timezone object, default None] Timezone to localize to.

pandas.Timestamp.replace

Timestamp.**replace**(*self*, *year=None*, *month=None*, *day=None*, *hour=None*, *minute=None*, *second=None*, *microsecond=None*, *nanosecond=None*, *tzinfo=<class 'object'>*, *fold=0*)

implements datetime.replace, handles nanoseconds.

Parameters

year [int, optional]

month [int, optional]

day [int, optional]

hour [int, optional]

minute [int, optional]

second [int, optional]

microsecond [int, optional]

nanosecond [int, optional]

tzinfo [tz-convertible, optional]

fold [int, optional, default is 0]

Returns

Timestamp with fields replaced

pandas.Timestamp.round

Timestamp.**round**(*self*, *freq*, *ambiguous='raise'*, *nonexistent='raise'*)

Round the Timestamp to the specified resolution.

Parameters

freq [str] Frequency string indicating the rounding resolution.

ambiguous [bool or {'raise', 'NaT'}, default 'raise'] The behavior is as follows:

- bool contains flags to determine if time is dst or not (note that this flag is only applicable for ambiguous fall dst dates).

- 'NaT' will return NaT for an ambiguous time.
- 'raise' will raise an AmbiguousTimeError for an ambiguous time.

New in version 0.24.0.

nonexistent [{'raise', 'shift_forward', 'shift_backward, 'NaT', timedelta}, default 'raise'] A nonexistent time does not exist in a particular timezone where clocks moved forward due to DST.

- 'shift_forward' will shift the nonexistent time forward to the closest existing time.
- 'shift_backward' will shift the nonexistent time backward to the closest existing time.
- 'NaT' will return NaT where there are nonexistent times.
- timedelta objects will shift nonexistent times by the timedelta.
- 'raise' will raise an NonExistentTimeError if there are nonexistent times.

New in version 0.24.0.

Returns

a new Timestamp rounded to the given resolution of *freq*

Raises

ValueError if the freq cannot be converted

pandas.Timestamp.strftime

`Timestamp.`**`strftime`**`()`
format -> strftime() style string.

pandas.Timestamp.strptime

`classmethod` `Timestamp.`**`strptime`**`(`*string*, *format*`)`
Function is not implemented. Use pd.to_datetime().

pandas.Timestamp.time

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

pandas.Timestamp.timestamp

Timestamp.**timestamp**()
Return POSIX timestamp as float.

pandas.Timestamp.timetuple

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

pandas.Timestamp.timetz

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

pandas.Timestamp.to_datetime64

Timestamp.**to_datetime64**()
Return a numpy.datetime64 object with 'ns' precision.

pandas.Timestamp.to_julian_date

Timestamp.**to_julian_date**(*self*)
Convert TimeStamp to a Julian Date. 0 Julian date is noon January 1, 4713 BC.

pandas.Timestamp.to_numpy

Timestamp.**to_numpy**()
Convert the Timestamp to a NumPy datetime64.

New in version 0.25.0.

This is an alias method for *Timestamp.to_datetime64()*. The dtype and copy parameters are available here only for compatibility. Their values will not affect the return value.

Returns

numpy.datetime64

See also:

DatetimeIndex.to_numpy Similar method for DatetimeIndex.

pandas.Timestamp.to_period

Timestamp.**to_period**(*self*, *freq=None*)
Return an period of which this timestamp is an observation.

pandas.Timestamp.to_pydatetime

Timestamp.**to_pydatetime**()
Convert a Timestamp object to a native Python datetime object.

If warn=True, issue a warning if nanoseconds is nonzero.

pandas.Timestamp.today

classmethod Timestamp.**today**(*cls*, *tz=None*)
Return the current time in the local timezone. This differs from datetime.today() in that it can be localized to a passed timezone.

Parameters

tz [str or timezone object, default None] Timezone to localize to.

pandas.Timestamp.toordinal

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

pandas.Timestamp.tz_convert

Timestamp.**tz_convert**(*self*, *tz*)
Convert tz-aware Timestamp to another time zone.

Parameters

tz [str, pytz.timezone, dateutil.tz.tzfile or None] Time zone for time which Timestamp will be converted to. None will remove timezone holding UTC time.

Returns

converted [Timestamp]

Raises

TypeError If Timestamp is tz-naive.

pandas.Timestamp.tz_localize

Timestamp.**tz_localize**(*self*, *tz*, *ambiguous='raise'*, *nonexistent='raise'*)

Convert naive Timestamp to local time zone, or remove timezone from tz-aware Timestamp.

Parameters

tz [str, pytz.timezone, dateutil.tz.tzfile or None] Time zone for time which Timestamp will be converted to. None will remove timezone holding local time.

ambiguous [bool, 'NaT', default 'raise'] When clocks moved backward due to DST, ambiguous times may arise. For example in Central European Time (UTC+01), when going from 03:00 DST to 02:00 non-DST, 02:30:00 local time occurs both at 00:30:00 UTC and at 01:30:00 UTC. In such a situation, the *ambiguous* parameter dictates how ambiguous times should be handled.

The behavior is as follows:

- bool contains flags to determine if time is dst or not (note that this flag is only applicable for ambiguous fall dst dates).
- 'NaT' will return NaT for an ambiguous time.
- 'raise' will raise an AmbiguousTimeError for an ambiguous time.

nonexistent ['shift_forward', 'shift_backward, 'NaT', timedelta, default 'raise'] A nonexistent time does not exist in a particular timezone where clocks moved forward due to DST.

The behavior is as follows:

- 'shift_forward' will shift the nonexistent time forward to the closest existing time.
- 'shift_backward' will shift the nonexistent time backward to the closest existing time.
- 'NaT' will return NaT where there are nonexistent times.
- timedelta objects will shift nonexistent times by the timedelta.
- 'raise' will raise an NonExistentTimeError if there are nonexistent times.

New in version 0.24.0.

Returns

localized [Timestamp]

Raises

TypeError If the Timestamp is tz-aware and tz is not None.

pandas.Timestamp.tzname

Timestamp.**tzname**()
Return self.tzinfo.tzname(self).

pandas.Timestamp.utcfromtimestamp

classmethod Timestamp.**utcfromtimestamp**(*ts*)
Construct a naive UTC datetime from a POSIX timestamp.

pandas.Timestamp.utcnow

classmethod Timestamp.**utcnow**()
Return a new Timestamp representing UTC day and time.

pandas.Timestamp.utcoffset

Timestamp.**utcoffset**()
Return self.tzinfo.utcoffset(self).

pandas.Timestamp.utctimetuple

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

pandas.Timestamp.weekday

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

isoformat	

Properties

Timestamp.asm8	Return numpy datetime64 format in nanoseconds.
Timestamp.day	
Timestamp.dayofweek	Return day of the week.
Timestamp.dayofyear	Return the day of the year.
Timestamp.days_in_month	Return the number of days in the month.
Timestamp.daysinmonth	Return the number of days in the month.
Timestamp.fold	
Timestamp.hour	
Timestamp.is_leap_year	Return True if year is a leap year.
Timestamp.is_month_end	Return True if date is last day of month.
Timestamp.is_month_start	Return True if date is first day of month.

continues on next page

Table 83 – continued from previous page

Timestamp.is_quarter_end	Return True if date is last day of the quarter.
Timestamp.is_quarter_start	Return True if date is first day of the quarter.
Timestamp.is_year_end	Return True if date is last day of the year.
Timestamp.is_year_start	Return True if date is first day of the year.
Timestamp.max	
Timestamp.microsecond	
Timestamp.min	
Timestamp.minute	
Timestamp.month	
Timestamp.nanosecond	
Timestamp.quarter	Return the quarter of the year.
Timestamp.resolution	
Timestamp.second	
Timestamp.tz	Alias for tzinfo.
Timestamp.tzinfo	
Timestamp.value	
Timestamp.week	Return the week number of the year.
Timestamp.weekofyear	Return the week number of the year.
Timestamp.year	

pandas.Timestamp.day

Timestamp.**day**

pandas.Timestamp.fold

Timestamp.**fold**

pandas.Timestamp.hour

Timestamp.**hour**

pandas.Timestamp.max

Timestamp.**max = Timestamp('2262-04-11 23:47:16.854775807')**

pandas.Timestamp.microsecond

Timestamp.**microsecond**

pandas.Timestamp.min

Timestamp.**min = Timestamp('1677-09-21 00:12:43.145225')**

pandas.Timestamp.minute

Timestamp.**minute**

pandas.Timestamp.month

Timestamp.**month**

pandas.Timestamp.nanosecond

Timestamp.**nanosecond**

pandas.Timestamp.resolution

Timestamp.**resolution = Timedelta('0 days 00:00:00.000000')**

pandas.Timestamp.second

Timestamp.**second**

pandas.Timestamp.tzinfo

Timestamp.**tzinfo**

pandas.Timestamp.value

Timestamp.**value**

pandas.Timestamp.year

Timestamp.**year**

Methods

Timestamp.astimezone(self, tz)	Convert tz-aware Timestamp to another time zone.
Timestamp.ceil(self, freq[, ambiguous, ...])	return a new Timestamp ceiled to this resolution.
Timestamp.combine(date, time)	date, time -> datetime with same date and time fields.
Timestamp.ctime()	Return ctime() style string.
Timestamp.date()	Return date object with same year, month and day.
Timestamp.day_name(self[, locale])	Return the day name of the Timestamp with specified locale.
Timestamp.dst()	Return self.tzinfo.dst(self).
Timestamp.floor(self, freq[, ambiguous, ...])	return a new Timestamp floored to this resolution.
Timestamp.freq	
Timestamp.freqstr	Return the total number of days in the month.
Timestamp.fromordinal(ordinal[, freq, tz])	Passed an ordinal, translate and convert to a ts.
Timestamp.fromtimestamp(ts)	timestamp[, tz] -> tz's local time from POSIX timestamp.
Timestamp.isocalendar()	Return a 3-tuple containing ISO year, week number, and weekday.
Timestamp.isoformat(self[, sep])	[sep] -> string in ISO 8601 format, YYYY-MM-DDT[HH[:MM[:SS[.mmm[uuu]]]]][+HH:MM].
Timestamp.isoweekday()	Return the day of the week represented by the date.
Timestamp.month_name(self[, locale])	Return the month name of the Timestamp with specified locale.
Timestamp.normalize(self)	Normalize Timestamp to midnight, preserving tz information.
Timestamp.now([tz])	Return new Timestamp object representing current time local to tz.
Timestamp.replace(self[, year, month, day, ...])	implements datetime.replace, handles nanoseconds.
Timestamp.round(self, freq[, ambiguous, ...])	Round the Timestamp to the specified resolution.
Timestamp.strftime()	format -> strftime() style string.
Timestamp.strptime(string, format)	Function is not implemented.
Timestamp.time()	Return time object with same time but with tzinfo=None.
Timestamp.timestamp()	Return POSIX timestamp as float.
Timestamp.timetuple()	Return time tuple, compatible with time.localtime().
Timestamp.timetz()	Return time object with same time and tzinfo.
Timestamp.to_datetime64()	Return a numpy.datetime64 object with 'ns' precision.
Timestamp.to_numpy()	Convert the Timestamp to a NumPy datetime64.
Timestamp.to_julian_date(self)	Convert TimeStamp to a Julian Date.
Timestamp.to_period(self[, freq])	Return an period of which this timestamp is an observation.
Timestamp.to_pydatetime()	Convert a Timestamp object to a native Python datetime object.
Timestamp.today(cls[, tz])	Return the current time in the local timezone.
Timestamp.toordinal()	Return proleptic Gregorian ordinal.
Timestamp.tz_convert(self, tz)	Convert tz-aware Timestamp to another time zone.
Timestamp.tz_localize(self, tz[, ambiguous, ...])	Convert naive Timestamp to local time zone, or remove timezone from tz-aware Timestamp.
Timestamp.tzname()	Return self.tzinfo.tzname(self).
Timestamp.utcfromtimestamp(ts)	Construct a naive UTC datetime from a POSIX timestamp.

continues on next page

Table 84 – continued from previous page

Timestamp.utcnow()	Return a new Timestamp representing UTC day and time.
Timestamp.utcoffset()	Return self.tzinfo.utcoffset(self).
Timestamp.utctimetuple()	Return UTC time tuple, compatible with time.localtime().
Timestamp.weekday()	Return the day of the week represented by the date.

pandas.Timestamp.freq

Timestamp.**freq**

pandas.Timestamp.isoformat

Timestamp.**isoformat**(*self*, *sep='T'*)

[sep] -> string in ISO 8601 format, YYYY-MM-DDT[HH[:MM[:SS[.mmm[uuu]]]]][+HH:MM]. sep is used to separate the year from the time, and defaults to 'T'. timespec specifies what components of the time to include (allowed values are 'auto', 'hours', 'minutes', 'seconds', 'milliseconds', and 'microseconds').

A collection of timestamps may be stored in a *arrays.DatetimeArray*. For timezone-aware data, the `.dtype` of a `DatetimeArray` is a *DatetimeTZDtype*. For timezone-naive data, `np.dtype("datetime64[ns]")` is used.

If the data are tz-aware, then every value in the array must have the same timezone.

arrays.DatetimeArray(values[, dtype, freq, copy])	Pandas ExtensionArray for tz-naive or tz-aware datetime data.

pandas.arrays.DatetimeArray

class pandas.arrays.**DatetimeArray**(*values*, *dtype=dtype('<M8[ns]')*, *freq=None*, *copy=False*)

Pandas ExtensionArray for tz-naive or tz-aware datetime data.

New in version 0.24.0.

> **Warning:** DatetimeArray is currently experimental, and its API may change without warning. In particular, `DatetimeArray.dtype` is expected to change to always be an instance of an `ExtensionDtype` subclass.

Parameters

values [Series, Index, DatetimeArray, ndarray] The datetime data.

For DatetimeArray *values* (or a Series or Index boxing one), *dtype* and *freq* will be extracted from *values*.

dtype [numpy.dtype or DatetimeTZDtype] Note that the only NumPy dtype allowed is 'datetime64[ns]'.

freq [str or Offset, optional] The frequency.

copy [bool, default False] Whether to copy the underlying array of values.

Attributes

None	

Methods

None	

DatetimeTZDtype([unit, tz])	An ExtensionDtype for timezone-aware datetime data.

pandas.DatetimeTZDtype

class pandas.**DatetimeTZDtype**(*unit='ns'*, *tz=None*)

An ExtensionDtype for timezone-aware datetime data.

This is not an actual numpy dtype, but a duck type.

Parameters

unit [str, default "ns"] The precision of the datetime data. Currently limited to `"ns"`.

tz [str, int, or datetime.tzinfo] The timezone.

Raises

pytz.UnknownTimeZoneError When the requested timezone cannot be found.

Examples

```
>>> pd.DatetimeTZDtype(tz='UTC')
datetime64[ns, UTC]
```

```
>>> pd.DatetimeTZDtype(tz='dateutil/US/Central')
datetime64[ns, tzfile('/usr/share/zoneinfo/US/Central')]
```

Attributes

unit	The precision of the datetime data.
tz	The timezone.

pandas.DatetimeTZDtype.unit

property DatetimeTZDtype.**unit**
The precision of the datetime data.

pandas.DatetimeTZDtype.tz

property DatetimeTZDtype.**tz**
The timezone.

Methods

None	

3.5.3 Timedelta data

NumPy can natively represent timedeltas. Pandas provides *Timedelta* for symmetry with *Timestamp*.

Timedelta([value, unit])	Represents a duration, the difference between two dates or times.

pandas.Timedelta

class pandas.**Timedelta**(*value=<object object>*, *unit=None*, ***kwargs*)
Represents a duration, the difference between two dates or times.

Timedelta is the pandas equivalent of python's `datetime.timedelta` and is interchangeable with it in most cases.

Parameters

value [Timedelta, timedelta, np.timedelta64, string, or integer]

unit [str, default 'ns'] Denote the unit of the input, if input is an integer.

Possible values:

- 'Y', 'M', 'W', 'D', 'T', 'S', 'L', 'U', or 'N'
- 'days' or 'day'
- 'hours', 'hour', 'hr', or 'h'
- 'minutes', 'minute', 'min', or 'm'
- 'seconds', 'second', or 'sec'
- 'milliseconds', 'millisecond', 'millis', or 'milli'
- 'microseconds', 'microsecond', 'micros', or 'micro'
- 'nanoseconds', 'nanosecond', 'nanos', 'nano', or 'ns'.

****kwargs** Available kwargs: {days, seconds, microseconds, milliseconds, minutes, hours, weeks}. Values for construction in compat with datetime.timedelta. Numpy ints and floats will be coerced to python ints and floats.

Notes

The `.value` attribute is always in ns.

Attributes

asm8	Return a numpy timedelta64 array scalar view.
components	Return a components namedtuple-like.
days	Number of days.
delta	Return the timedelta in nanoseconds (ns), for internal compatibility.
microseconds	Number of microseconds (>= 0 and less than 1 second).
nanoseconds	Return the number of nanoseconds (n), where 0 <= n < 1 microsecond.
resolution_string	Return a string representing the lowest timedelta resolution.
seconds	Number of seconds (>= 0 and less than 1 day).

pandas.Timedelta.asm8

Timedelta.**asm8**

Return a numpy timedelta64 array scalar view.

Provides access to the array scalar view (i.e. a combination of the value and the units) associated with the numpy.timedelta64().view(), including a 64-bit integer representation of the timedelta in nanoseconds (Python int compatible).

Returns

numpy timedelta64 array scalar view Array scalar view of the timedelta in nanoseconds.

Examples

```
>>> td = pd.Timedelta('1 days 2 min 3 us 42 ns')
>>> td.asm8
numpy.timedelta64(86520000003042,'ns')
```

```
>>> td = pd.Timedelta('2 min 3 s')
>>> td.asm8
numpy.timedelta64(123000000000,'ns')
```

```
>>> td = pd.Timedelta('3 ms 5 us')
>>> td.asm8
numpy.timedelta64(3005000,'ns')
```

```
>>> td = pd.Timedelta(42, unit='ns')
>>> td.asm8
numpy.timedelta64(42,'ns')
```