pandas.Timedelta.components

Timedelta.**components**
Return a components namedtuple-like.

pandas.Timedelta.days

Timedelta.**days**
Number of days.

pandas.Timedelta.delta

Timedelta.**delta**
Return the timedelta in nanoseconds (ns), for internal compatibility.

Returns

int Timedelta in nanoseconds.

Examples

```
>>> td = pd.Timedelta('1 days 42 ns')
>>> td.delta
86400000000042
```

```
>>> td = pd.Timedelta('3 s')
>>> td.delta
3000000000
```

```
>>> td = pd.Timedelta('3 ms 5 us')
>>> td.delta
3005000
```

```
>>> td = pd.Timedelta(42, unit='ns')
>>> td.delta
42
```

pandas.Timedelta.microseconds

Timedelta.**microseconds**
Number of microseconds (>= 0 and less than 1 second).

pandas.Timedelta.nanoseconds

Timedelta.**nanoseconds**

Return the number of nanoseconds (n), where 0 <= n < 1 microsecond.

Returns

int Number of nanoseconds.

See also:

Timedelta.components Return all attributes with assigned values (i.e. days, hours, minutes, seconds, milliseconds, microseconds, nanoseconds).

Examples

Using string input

```
>>> td = pd.Timedelta('1 days 2 min 3 us 42 ns')
>>> td.nanoseconds
42
```

Using integer input

```
>>> td = pd.Timedelta(42, unit='ns')
>>> td.nanoseconds
42
```

pandas.Timedelta.resolution_string

Timedelta.**resolution_string**

Return a string representing the lowest timedelta resolution.

Each timedelta has a defined resolution that represents the lowest OR most granular level of precision. Each level of resolution is represented by a short string as defined below:

Resolution: Return value

- Days: 'D'
- Hours: 'H'
- Minutes: 'T'
- Seconds: 'S'
- Milliseconds: 'L'
- Microseconds: 'U'
- Nanoseconds: 'N'

Returns

str Timedelta resolution.

Examples

```
>>> td = pd.Timedelta('1 days 2 min 3 us 42 ns')
>>> td.resolution
'N'
```

```
>>> td = pd.Timedelta('1 days 2 min 3 us')
>>> td.resolution
'U'
```

```
>>> td = pd.Timedelta('2 min 3 s')
>>> td.resolution
'S'
```

```
>>> td = pd.Timedelta(36, unit='us')
>>> td.resolution
'U'
```

pandas.Timedelta.seconds

`Timedelta.`**`seconds`**

Number of seconds (>= 0 and less than 1 day).

freq	
is_populated	
value	

Methods

ceil(self, freq)	Return a new Timedelta ceiled to this resolution.
floor(self, freq)	Return a new Timedelta floored to this resolution.
isoformat()	Format Timedelta as ISO 8601 Duration like `P[n]Y[n]M[n]DT[n]H[n]M[n]S`, where the `[n]` s are replaced by the values.
round(self, freq)	Round the Timedelta to the specified resolution.
to_numpy()	Convert the Timedelta to a NumPy timedelta64.
to_pytimedelta()	Convert a pandas Timedelta object into a python timedelta object.
to_timedelta64()	Return a numpy.timedelta64 object with 'ns' precision.
total_seconds()	Total duration of timedelta in seconds (to ns precision).
view()	Array view compatibility.

pandas.Timedelta.ceil

Timedelta.**ceil**(*self*, *freq*)

Return a new Timedelta ceiled to this resolution.

Parameters

freq [str] Frequency string indicating the ceiling resolution.

pandas.Timedelta.floor

Timedelta.**floor**(*self*, *freq*)

Return a new Timedelta floored to this resolution.

Parameters

freq [str] Frequency string indicating the flooring resolution.

pandas.Timedelta.isoformat

Timedelta.**isoformat**()

Format Timedelta as ISO 8601 Duration like `P[n]Y[n]M[n]DT[n]H[n]M[n]S`, where the `[n]` s are replaced by the values. See https://en.wikipedia.org/wiki/ISO_8601#Durations.

Returns

formatted [str]

See also:

Timestamp.isoformat

Notes

The longest component is days, whose value may be larger than 365. Every component is always included, even if its value is 0. Pandas uses nanosecond precision, so up to 9 decimal places may be included in the seconds component. Trailing 0's are removed from the seconds component after the decimal. We do not 0 pad components, so it's . . . *T5H*. . . , not . . . *T05H*. . .

Examples

```
>>> td = pd.Timedelta(days=6, minutes=50, seconds=3,
...                   milliseconds=10, microseconds=10, nanoseconds=12)
>>> td.isoformat()
'P6DT0H50M3.010010012S'
>>> pd.Timedelta(hours=1, seconds=10).isoformat()
'P0DT0H0M10S'
>>> pd.Timedelta(hours=1, seconds=10).isoformat()
'P0DT0H0M10S'
>>> pd.Timedelta(days=500.5).isoformat()
'P500DT12H0MS'
```

pandas.Timedelta.round

Timedelta.**round**(*self*, *freq*)

Round the Timedelta to the specified resolution.

Parameters

freq [str] Frequency string indicating the rounding resolution.

Returns

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

Raises

ValueError if the freq cannot be converted

pandas.Timedelta.to_numpy

Timedelta.**to_numpy**()

Convert the Timedelta to a NumPy timedelta64.

New in version 0.25.0.

This is an alias method for *Timedelta.to_timedelta64()*. The dtype and copy parameters are available here only for compatibility. Their values will not affect the return value.

Returns

numpy.timedelta64

See also:

Series.to_numpy Similar method for Series.

pandas.Timedelta.to_pytimedelta

Timedelta.**to_pytimedelta**()

Convert a pandas Timedelta object into a python timedelta object.

Timedelta objects are internally saved as numpy datetime64[ns] dtype. Use to_pytimedelta() to convert to object dtype.

Returns

datetime.timedelta or numpy.array of datetime.timedelta

See also:

to_timedelta Convert argument to Timedelta type.

Notes

Any nanosecond resolution will be lost.

pandas.Timedelta.to_timedelta64

Timedelta.**to_timedelta64**()
Return a numpy.timedelta64 object with 'ns' precision.

pandas.Timedelta.total_seconds

Timedelta.**total_seconds**()
Total duration of timedelta in seconds (to ns precision).

pandas.Timedelta.view

Timedelta.**view**()
Array view compatibility.

Properties

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

pandas.Timedelta.freq

Timedelta.**freq**

pandas.Timedelta.is_populated

Timedelta.**is_populated**

pandas.Timedelta.max

Timedelta.**max = Timedelta('106751 days 23:47:16.854775')**

pandas.Timedelta.min

Timedelta.**min = Timedelta('-106752 days +00:12:43.145224')**

pandas.Timedelta.resolution

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

pandas.Timedelta.value

Timedelta.**value**

Methods

Timedelta.ceil(self, freq)	Return a new Timedelta ceiled to this resolution.
Timedelta.floor(self, freq)	Return a new Timedelta floored to this resolution.
Timedelta.isoformat()	Format Timedelta as ISO 8601 Duration like `P[n]Y[n]M[n]DT[n]H[n]M[n]S`, where the `[n]` s are replaced by the values.
Timedelta.round(self, freq)	Round the Timedelta to the specified resolution.
Timedelta.to_pytimedelta()	Convert a pandas Timedelta object into a python timedelta object.
Timedelta.to_timedelta64()	Return a numpy.timedelta64 object with 'ns' precision.
Timedelta.to_numpy()	Convert the Timedelta to a NumPy timedelta64.
Timedelta.total_seconds()	Total duration of timedelta in seconds (to ns precision).

A collection of timedeltas may be stored in a `TimedeltaArray`.

arrays.TimedeltaArray(values[, dtype, freq, ...])	Pandas ExtensionArray for timedelta data.

pandas.arrays.TimedeltaArray

class pandas.arrays.**TimedeltaArray**(*values*, *dtype=dtype('<m8[ns]')*, *freq=None*, *copy=False*)

Pandas ExtensionArray for timedelta data.

New in version 0.24.0.

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

Parameters

values [array-like] The timedelta data.

dtype [numpy.dtype] Currently, only `numpy.dtype("timedelta64[ns]")` is accepted.

freq [Offset, optional]

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

Attributes

None	

Methods

None	

3.5.4 Timespan data

Pandas represents spans of times as *Period* objects.

3.5.5 Period

Period([value, freq, ordinal, year, month, ...])	Represents a period of time.

pandas.Period

class pandas.**Period**(*value=None*, *freq=None*, *ordinal=None*, *year=None*, *month=None*, *quarter=None*, *day=None*, *hour=None*, *minute=None*, *second=None*)

Represents a period of time.

Parameters

value [Period or str, default None] The time period represented (e.g., '4Q2005').

freq [str, default None] One of pandas period strings or corresponding objects.

ordinal [int, default None] The period offset from the gregorian proleptic epoch.

year [int, default None] Year value of the period.

month [int, default 1] Month value of the period.

quarter [int, default None] Quarter value of the period.

day [int, default 1] Day value of the period.

hour [int, default 0] Hour value of the period.

minute [int, default 0] Minute value of the period.

second [int, default 0] Second value of the period.

Attributes

day	Get day of the month that a Period falls on.
dayofweek	Day of the week the period lies in, with Monday=0 and Sunday=6.
dayofyear	Return the day of the year.
days_in_month	Get the total number of days in the month that this period falls on.
daysinmonth	Get the total number of days of the month that the Period falls in.
hour	Get the hour of the day component of the Period.
minute	Get minute of the hour component of the Period.
qyear	Fiscal year the Period lies in according to its starting-quarter.
second	Get the second component of the Period.
start_time	Get the Timestamp for the start of the period.
week	Get the week of the year on the given Period.
weekday	Day of the week the period lies in, with Monday=0 and Sunday=6.

pandas.Period.day

Period.**day**

Get day of the month that a Period falls on.

Returns

int

See also:

Period.dayofweek Get the day of the week.

Period.dayofyear Get the day of the year.

Examples

```
>>> p = pd.Period("2018-03-11", freq='H')
>>> p.day
11
```

pandas.Period.dayofweek

Period.**dayofweek**

Day of the week the period lies in, with Monday=0 and Sunday=6.

If the period frequency is lower than daily (e.g. hourly), and the period spans over multiple days, the day at the start of the period is used.

If the frequency is higher than daily (e.g. monthly), the last day of the period is used.

Returns

int Day of the week.

See also:

Period.dayofweek Day of the week the period lies in.

Period.weekday Alias of Period.dayofweek.

Period.day Day of the month.

Period.dayofyear Day of the year.

Examples

```
>>> per = pd.Period('2017-12-31 22:00', 'H')
>>> per.dayofweek
6
```

For periods that span over multiple days, the day at the beginning of the period is returned.

```
>>> per = pd.Period('2017-12-31 22:00', '4H')
>>> per.dayofweek
6
>>> per.start_time.dayofweek
6
```

For periods with a frequency higher than days, the last day of the period is returned.

```
>>> per = pd.Period('2018-01', 'M')
>>> per.dayofweek
2
>>> per.end_time.dayofweek
2
```

pandas.Period.dayofyear

Period.**dayofyear**

Return the day of the year.

This attribute returns the day of the year on which the particular date occurs. The return value ranges between 1 to 365 for regular years and 1 to 366 for leap years.

Returns

int The day of year.

See also:

Period.day Return the day of the month.

Period.dayofweek Return the day of week.

PeriodIndex.dayofyear Return the day of year of all indexes.

Examples

```
>>> period = pd.Period("2015-10-23", freq='H')
>>> period.dayofyear
296
>>> period = pd.Period("2012-12-31", freq='D')
>>> period.dayofyear
366
>>> period = pd.Period("2013-01-01", freq='D')
>>> period.dayofyear
1
```

pandas.Period.days_in_month

Period.**days_in_month**

Get the total number of days in the month that this period falls on.

Returns

int

See also:

Period.daysinmonth Gets the number of days in the month.

DatetimeIndex.daysinmonth Gets the number of days in the month.

calendar.monthrange Returns a tuple containing weekday (0-6 ~ Mon-Sun) and number of days (28-31).

Examples

```
>>> p = pd.Period('2018-2-17')
>>> p.days_in_month
28
```

```
>>> pd.Period('2018-03-01').days_in_month
31
```

Handles the leap year case as well:

```
>>> p = pd.Period('2016-2-17')
>>> p.days_in_month
29
```

pandas.Period.daysinmonth

Period.**daysinmonth**

Get the total number of days of the month that the Period falls in.

Returns

int

See also:

Period.days_in_month Return the days of the month.

Period.dayofyear Return the day of the year.

Examples

```
>>> p = pd.Period("2018-03-11", freq='H')
>>> p.daysinmonth
31
```

pandas.Period.hour

Period.**hour**

Get the hour of the day component of the Period.

Returns

int The hour as an integer, between 0 and 23.

See also:

Period.second Get the second component of the Period.

Period.minute Get the minute component of the Period.

Examples

```
>>> p = pd.Period("2018-03-11 13:03:12.050000")
>>> p.hour
13
```

Period longer than a day

```
>>> p = pd.Period("2018-03-11", freq="M")
>>> p.hour
0
```

pandas.Period.minute

Period.**minute**

Get minute of the hour component of the Period.

Returns

int The minute as an integer, between 0 and 59.

See also:

Period.hour Get the hour component of the Period.

Period.second Get the second component of the Period.

Examples

```
>>> p = pd.Period("2018-03-11 13:03:12.050000")
>>> p.minute
3
```

pandas.Period.qyear

Period.**qyear**

Fiscal year the Period lies in according to its starting-quarter.

The *year* and the *qyear* of the period will be the same if the fiscal and calendar years are the same. When they are not, the fiscal year can be different from the calendar year of the period.

Returns

int The fiscal year of the period.

See also:

Period.year Return the calendar year of the period.

Examples

If the natural and fiscal year are the same, *qyear* and *year* will be the same.

```
>>> per = pd.Period('2018Q1', freq='Q')
>>> per.qyear
2018
>>> per.year
2018
```

If the fiscal year starts in April (*Q-MAR*), the first quarter of 2018 will start in April 2017. *year* will then be 2018, but *qyear* will be the fiscal year, 2018.

```
>>> per = pd.Period('2018Q1', freq='Q-MAR')
>>> per.start_time
Timestamp('2017-04-01 00:00:00')
>>> per.qyear
2018
>>> per.year
2017
```

pandas.Period.second

Period.**second**

Get the second component of the Period.

Returns

int The second of the Period (ranges from 0 to 59).

See also:

Period.hour Get the hour component of the Period.

Period.minute Get the minute component of the Period.

Examples

```
>>> p = pd.Period("2018-03-11 13:03:12.050000")
>>> p.second
12
```

pandas.Period.start_time

Period.**start_time**

Get the Timestamp for the start of the period.

Returns

Timestamp

See also:

Period.end_time Return the end Timestamp.

Period.dayofyear Return the day of year.

Period.daysinmonth Return the days in that month.

Period.dayofweek Return the day of the week.

Examples

```
>>> period = pd.Period('2012-1-1', freq='D')
>>> period
Period('2012-01-01', 'D')
```

```
>>> period.start_time
Timestamp('2012-01-01 00:00:00')
```

```
>>> period.end_time
Timestamp('2012-01-01 23:59:59.999999999')
```

pandas.Period.week

Period.**week**

Get the week of the year on the given Period.

Returns

int

See also:

Period.dayofweek Get the day component of the Period.

Period.weekday Get the day component of the Period.

Examples

```
>>> p = pd.Period("2018-03-11", "H")
>>> p.week
10
```

```
>>> p = pd.Period("2018-02-01", "D")
>>> p.week
5
```

```
>>> p = pd.Period("2018-01-06", "D")
>>> p.week
1
```

pandas.Period.weekday

Period.**weekday**

Day of the week the period lies in, with Monday=0 and Sunday=6.

If the period frequency is lower than daily (e.g. hourly), and the period spans over multiple days, the day at the start of the period is used.

If the frequency is higher than daily (e.g. monthly), the last day of the period is used.

Returns

int Day of the week.

See also:

Period.dayofweek Day of the week the period lies in.

Period.weekday Alias of Period.dayofweek.

Period.day Day of the month.

Period.dayofyear Day of the year.

Examples

```
>>> per = pd.Period('2017-12-31 22:00', 'H')
>>> per.dayofweek
6
```

For periods that span over multiple days, the day at the beginning of the period is returned.

```
>>> per = pd.Period('2017-12-31 22:00', '4H')
>>> per.dayofweek
6
>>> per.start_time.dayofweek
6
```

For periods with a frequency higher than days, the last day of the period is returned.

```
>>> per = pd.Period('2018-01', 'M')
>>> per.dayofweek
2
>>> per.end_time.dayofweek
2
```

end_time	
freq	
freqstr	
is_leap_year	
month	
ordinal	
quarter	
weekofyear	
year	

Methods

asfreq()	Convert Period to desired frequency, at the start or end of the interval.
strftime()	Returns the string representation of the *Period*, depending on the selected `fmt`.
to_timestamp()	Return the Timestamp representation of the Period.

pandas.Period.asfreq

`Period.`**`asfreq`**`()`

Convert Period to desired frequency, at the start or end of the interval.

Parameters

freq [str] The desired frequency.

how [{'E', 'S', 'end', 'start'}, default 'end'] Start or end of the timespan.

Returns

resampled [Period]

pandas.Period.strftime

`Period.`**`strftime`**`()`

Returns the string representation of the *Period*, depending on the selected `fmt`. `fmt` must be a string containing one or several directives. The method recognizes the same directives as the `time.strftime()` function of the standard Python distribution, as well as the specific additional directives `%f`, `%F`, `%q`. (formatting & docs originally from scikits.timeries).

Directive	Meaning	Notes
%a	Locale's abbreviated weekday name.	
%A	Locale's full weekday name.	
%b	Locale's abbreviated month name.	
%B	Locale's full month name.	
%c	Locale's appropriate date and time representation.	
%d	Day of the month as a decimal number [01,31].	
%f	'Fiscal' year without a century as a decimal number [00,99]	(1)
%F	'Fiscal' year with a century as a decimal number	(2)
%H	Hour (24-hour clock) as a decimal number [00,23].	
%I	Hour (12-hour clock) as a decimal number [01,12].	
%j	Day of the year as a decimal number [001,366].	
%m	Month as a decimal number [01,12].	
%M	Minute as a decimal number [00,59].	
%p	Locale's equivalent of either AM or PM.	(3)
%q	Quarter as a decimal number [01,04]	
%S	Second as a decimal number [00,61].	(4)
%U	Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0.	(5)
%w	Weekday as a decimal number [0(Sunday),6].	
%W	Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.	(5)
%x	Locale's appropriate date representation.	
%X	Locale's appropriate time representation.	
%y	Year without century as a decimal number [00,99].	
%Y	Year with century as a decimal number.	
%Z	Time zone name (no characters if no time zone exists).	
%%	A literal '%' character.	

Notes

(1) The %f directive is the same as %y if the frequency is not quarterly. Otherwise, it corresponds to the 'fiscal' year, as defined by the *qyear* attribute.

(2) The %F directive is the same as %Y if the frequency is not quarterly. Otherwise, it corresponds to the 'fiscal' year, as defined by the *qyear* attribute.

(3) The %p directive only affects the output hour field if the %I directive is used to parse the hour.

(4) The range really is 0 to 61; this accounts for leap seconds and the (very rare) double leap seconds.

(5) The %U and %W directives are only used in calculations when the day of the week and the year are specified.

Examples

```
>>> a = Period(freq='Q-JUL', year=2006, quarter=1)
>>> a.strftime('%F-Q%q')
'2006-Q1'
>>> # Output the last month in the quarter of this date
>>> a.strftime('%b-%Y')
'Oct-2005'
>>>
>>> a = Period(freq='D', year=2001, month=1, day=1)
>>> a.strftime('%d-%b-%Y')
'01-Jan-2006'
>>> a.strftime('%b. %d, %Y was a %A')
'Jan. 01, 2001 was a Monday'
```

pandas.Period.to_timestamp

Period.**to_timestamp**()

Return the Timestamp representation of the Period.

Uses the target frequency specified at the part of the period specified by *how*, which is either *Start* or *Finish*.

Parameters

freq [str or DateOffset] Target frequency. Default is 'D' if self.freq is week or longer and 'S' otherwise.

how [str, default 'S' (start)] One of 'S', 'E'. Can be aliased as case insensitive 'Start', 'Finish', 'Begin', 'End'.

Returns

Timestamp

now	

Properties

Period.day	Get day of the month that a Period falls on.
Period.dayofweek	Day of the week the period lies in, with Monday=0 and Sunday=6.
Period.dayofyear	Return the day of the year.
Period.days_in_month	Get the total number of days in the month that this period falls on.
Period.daysinmonth	Get the total number of days of the month that the Period falls in.
Period.end_time	
Period.freq	
Period.freqstr	
Period.hour	Get the hour of the day component of the Period.
Period.is_leap_year	
Period.minute	Get minute of the hour component of the Period.

continues on next page

Table 97 – continued from previous page

Period.month	
Period.ordinal	
Period.quarter	
Period.qyear	Fiscal year the Period lies in according to its starting-quarter.
Period.second	Get the second component of the Period.
Period.start_time	Get the Timestamp for the start of the period.
Period.week	Get the week of the year on the given Period.
Period.weekday	Day of the week the period lies in, with Monday=0 and Sunday=6.
Period.weekofyear	
Period.year	

pandas.Period.end_time

Period.**end_time**

pandas.Period.freq

Period.**freq**

pandas.Period.freqstr

Period.**freqstr**

pandas.Period.is_leap_year

Period.**is_leap_year**

pandas.Period.month

Period.**month**

pandas.Period.ordinal

Period.**ordinal**

pandas.Period.quarter

Period.**quarter**

pandas.Period.weekofyear

Period.**weekofyear**

pandas.Period.year

Period.**year**

Methods

Period.asfreq()	Convert Period to desired frequency, at the start or end of the interval.
Period.now()	
Period.strftime()	Returns the string representation of the *Period*, depending on the selected `fmt`.
Period.to_timestamp()	Return the Timestamp representation of the Period.

pandas.Period.now

Period.**now**()

A collection of timedeltas may be stored in a *arrays.PeriodArray*. Every period in a `PeriodArray` must have the same `freq`.

arrays.PeriodArray(values[, freq, dtype, copy])	Pandas ExtensionArray for storing Period data.

pandas.arrays.PeriodArray

class pandas.arrays.**PeriodArray**(*values*, *freq=None*, *dtype=None*, *copy=False*)

Pandas ExtensionArray for storing Period data.

Users should use `period_array()` to create new instances.

Parameters

values [Union[PeriodArray, Series[period], ndarray[int], PeriodIndex]] The data to store. These should be arrays that can be directly converted to ordinals without inference or copy (PeriodArray, ndarray[int64]), or a box around such an array (Series[period], PeriodIndex).

freq [str or DateOffset] The *freq* to use for the array. Mostly applicable when *values* is an ndarray of integers, when *freq* is required. When *values* is a PeriodArray (or box around), it's checked that `values.freq` matches *freq*.

dtype [PeriodDtype, optional] A PeriodDtype instance from which to extract a *freq*. If both *freq* and *dtype* are specified, then the frequencies must match.

copy [bool, default False] Whether to copy the ordinals before storing.

See also:

period_array Create a new PeriodArray.
PeriodIndex Immutable Index for period data.

Notes

There are two components to a PeriodArray

- ordinals : integer ndarray
- freq : pd.tseries.offsets.Offset

The values are physically stored as a 1-D ndarray of integers. These are called "ordinals" and represent some kind of offset from a base.

The *freq* indicates the span covered by each element of the array. All elements in the PeriodArray have the same *freq*.

Attributes

None	

Methods

None	

PeriodDtype([freq])	An ExtensionDtype for Period data.

pandas.PeriodDtype

class pandas.**PeriodDtype**(*freq=None*)

An ExtensionDtype for Period data.

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

Parameters

freq [str or DateOffset] The frequency of this PeriodDtype.

Examples

```
>>> pd.PeriodDtype(freq='D')
period[D]
```

```
>>> pd.PeriodDtype(freq=pd.offsets.MonthEnd())
period[M]
```

Attributes

freq	The frequency object of this PeriodDtype.

pandas.PeriodDtype.freq

property PeriodDtype.**freq**
The frequency object of this PeriodDtype.

Methods

None	

3.5.6 Interval data

Arbitrary intervals can be represented as *Interval* objects.

Interval	Immutable object implementing an Interval, a bounded slice-like interval.

pandas.Interval

class pandas.**Interval**
Immutable object implementing an Interval, a bounded slice-like interval.

Parameters

left [orderable scalar] Left bound for the interval.

right [orderable scalar] Right bound for the interval.

closed [{'right', 'left', 'both', 'neither'}, default 'right'] Whether the interval is closed on the left-side, right-side, both or neither. See the Notes for more detailed explanation.

See also:

IntervalIndex An Index of Interval objects that are all closed on the same side.
cut Convert continuous data into discrete bins (Categorical of Interval objects).
qcut Convert continuous data into bins (Categorical of Interval objects) based on quantiles.
Period Represents a period of time.

Notes

The parameters *left* and *right* must be from the same type, you must be able to compare them and they must satisfy `left <= right`.

A closed interval (in mathematics denoted by square brackets) contains its endpoints, i.e. the closed interval `[0, 5]` is characterized by the conditions `0 <= x <= 5`. This is what `closed='both'` stands for. An open interval (in mathematics denoted by parentheses) does not contain its endpoints, i.e. the open interval `(0, 5)` is characterized by the conditions `0 < x < 5`. This is what `closed='neither'` stands for. Intervals can also be half-open or half-closed, i.e. `[0, 5)` is described by `0 <= x < 5` (`closed='left'`) and `(0, 5]` is described by `0 < x <= 5` (`closed='right'`).

Examples

It is possible to build Intervals of different types, like numeric ones:

```
>>> iv = pd.Interval(left=0, right=5)
>>> iv
Interval(0, 5, closed='right')
```

You can check if an element belongs to it

```
>>> 2.5 in iv
True
```

You can test the bounds (`closed='right'`, so `0 < x <= 5`):

```
>>> 0 in iv
False
>>> 5 in iv
True
>>> 0.0001 in iv
True
```

Calculate its length

```
>>> iv.length
5
```

You can operate with + and * over an Interval and the operation is applied to each of its bounds, so the result depends on the type of the bound elements

```
>>> shifted_iv = iv + 3
>>> shifted_iv
Interval(3, 8, closed='right')
>>> extended_iv = iv * 10.0
>>> extended_iv
Interval(0.0, 50.0, closed='right')
```

To create a time interval you can use Timestamps as the bounds

```
>>> year_2017 = pd.Interval(pd.Timestamp('2017-01-01 00:00:00'),
...                         pd.Timestamp('2018-01-01 00:00:00'),
...                         closed='left')
>>> pd.Timestamp('2017-01-01 00:00') in year_2017
True
>>> year_2017.length
Timedelta('365 days 00:00:00')
```

And also you can create string intervals

```
>>> volume_1 = pd.Interval('Ant', 'Dog', closed='both')
>>> 'Bee' in volume_1
True
```

Attributes

closed	Whether the interval is closed on the left-side, right-side, both or neither.
closed_left	Check if the interval is closed on the left side.
closed_right	Check if the interval is closed on the right side.
is_empty	Indicates if an interval is empty, meaning it contains no points.
left	Left bound for the interval.
length	Return the length of the Interval.
mid	Return the midpoint of the Interval.
open_left	Check if the interval is open on the left side.
open_right	Check if the interval is open on the right side.
right	Right bound for the interval.

pandas.Interval.closed

Interval.**closed**
Whether the interval is closed on the left-side, right-side, both or neither.

pandas.Interval.closed_left

Interval.**closed_left**
Check if the interval is closed on the left side.

For the meaning of *closed* and *open* see *Interval*.

Returns

bool True if the Interval is closed on the left-side.

pandas.Interval.closed_right

Interval.**closed_right**
Check if the interval is closed on the right side.

For the meaning of *closed* and *open* see *Interval*.

Returns

bool True if the Interval is closed on the left-side.

pandas.Interval.is_empty

Interval.**is_empty**
Indicates if an interval is empty, meaning it contains no points.

New in version 0.25.0.

Returns