Types in datetime module

date - stores calendar date(year,month,day) using the Gregorian calendar time - stores time of day as hours,minutes,seconds and microseconds datetime - stores both date and time timedelta - represents difference between two datetime values as days,seconds and microseconds tzinfo - base type for storing time zone information

In [2]:
```
#datetime stores both date and time down to the microsecond

from datetime import datetime
now=datetime.now()
now
print(now)
print(now.year,now.month,now.day)
```

```
2023-01-20 10:27:06.715178
2023 1 20
```

In [5]:
```
#difference between dates
delta=datetime(2011,1,7)-datetime(2008,6,24,8,15)
print(delta)
print(delta.days)
print(delta.seconds)
```

```
926 days, 15:45:00
926
56700
```

In [6]:
```
#timedelta represents the temporal difference between two datetime objects
# Add or subtract a timedelta object or multiple thereof to a datetime object
#to yield a new shifted object

from datetime import timedelta
start=datetime(2023,1,16)
print(start+timedelta(12))
print(start-2*timedelta(4))
```

```
2023-01-28 00:00:00
2023-01-08 00:00:00
```

In [8]:
```
#converting between string and datetime
#

s=datetime(2023,1,16)
print(str(s))
#print(type(s))
print(s.strftime('%y-%m-%d'))
t='2020-11-26'
print(datetime.strptime(t,'%Y-%m-%d'))
```

```
2023-01-16 00:00:00
23-01-16
2020-11-26 00:00:00
```

In [ ]:
```
dates=['7/6/2011','8/6/2011']
[datetime.strptime(x,'%m/%d/%Y') for x in dates]
```

In [ ]:
```
#datetime.strptime is a good way to parse a date with a known format. but we need to
#write a format spec each time, especially for common date formats.

#use parser.parse method in the third-partydateutil package

from dateutil.parser import parse
parse('2011-01-03')
```

In [11]:
```
#dateutil is capable of parsing most human-intelligible date representations
from dateutil.parser import parse
parse('Jan 31, 1997 8:30 PM')

#when day appears first
parse('6/12/2011',dayfirst=True)
```

Out[11]:
```
datetime.datetime(2011, 12, 6, 0, 0)
```

In [ ]:
```
#to_datetime method parses many different kinds of date representations,  NaT means Not

import pandas as pd
dates=['2011-06-09','2011-08-23']
pd.to_datetime(dates)
idx=pd.to_datetime(dates+[None])
idx
idx[2]
```

In [13]:
```
#Time Series basics
import pandas as pd
from datetime import datetime
import numpy as np
dates = [datetime(2011, 1, 2), datetime(2011, 1, 5), datetime(2011, 1, 7),
  datetime(2011, 1, 8), datetime(2011, 1, 10), datetime(2011, 1, 12)]
ts = pd.Series(np.random.randn(6), index=dates)
print(ts)
print(ts.index)
print(type(ts))
```

```
2011-01-02    0.490931
2011-01-05   -0.179752
2011-01-07   -0.606554
2011-01-08   -0.496322
2011-01-10   -1.782412
2011-01-12   -0.937306
dtype: float64
DatetimeIndex(['2011-01-02', '2011-01-05', '2011-01-07', '2011-01-08',
               '2011-01-10', '2011-01-12'],
              dtype='datetime64[ns]', freq=None)
<class 'pandas.core.series.Series'>
```

In [ ]:
```
#Scalar values from a DatetimeIndex are pandas Timestamp objects
stamp = ts.index[0]
stamp

#A Timestamp can be substituted anywhere you would use a datetime object
```

In [ ]:
```
#Indexing, Selection, Subsetting
#TimeSeries is a subclass of Series and thus behaves in the same way with regard to
#indexing and selecting data based on label:
stamp = ts.index[2]
print(ts[stamp])
#As a convenience, you can also pass a string that is interpretable as a date:
print(ts['1/10/2011'])
print(ts['20110110'])
```

In [15]:
```
 #For longer time series, a year or only a year and month can be passed
#to easily select slices of data:

longer_ts = pd.Series(np.random.randn(1000),
                      index=pd.date_range('1/1/2000', periods=1000))
print(longer_ts)
print(longer_ts['2001'])
print(longer_ts['2001-05'])
```

```
2000-01-01    0.262730
2000-01-02    2.555813
2000-01-03   -1.585208
2000-01-04    2.143942
2000-01-05   -0.780031
                ...
2002-09-22    1.068814
2002-09-23   -0.914543
2002-09-24    0.359299
2002-09-25   -0.069370
2002-09-26    0.044389
Freq: D, Length: 1000, dtype: float64
2001-01-01    2.450036
2001-01-02    0.739998
2001-01-03   -0.778526
2001-01-04    0.432014
2001-01-05   -1.541915
                ...
2001-12-27   -0.090495
2001-12-28    1.565238
2001-12-29   -1.100152
2001-12-30   -1.155328
2001-12-31    1.243373
Freq: D, Length: 365, dtype: float64
2001-05-01    0.045566
2001-05-02    0.992080
2001-05-03   -0.062243
2001-05-04   -1.328534
2001-05-05   -0.266630
2001-05-06   -0.059270
2001-05-07   -1.009743
2001-05-08   -0.987349
```

```
2001-05-09    0.972626
2001-05-10   -0.019313
2001-05-11    0.567745
2001-05-12    0.264697
2001-05-13    0.560580
2001-05-14   -0.531430
2001-05-15    1.496370
2001-05-16    1.411320
2001-05-17    0.486609
2001-05-18    0.674990
2001-05-19   -2.684320
2001-05-20    0.091022
2001-05-21    0.488894
2001-05-22   -0.081412
2001-05-23    1.871787
2001-05-24    1.332545
2001-05-25   -0.567973
2001-05-26   -0.974575
2001-05-27   -0.515048
2001-05-28    0.326682
2001-05-29    0.788454
2001-05-30    0.498072
2001-05-31   -0.046288
Freq: D, dtype: float64
```

In [ ]:
```
#Slicing with dates works just like with a regular Series:
ts[datetime(2011, 1, 7):]
```

In [ ]:
```
#Because most time series data is ordered chronologically, you can slice
#with timestamps not contained in a time series to perform a range query
ts['1/6/2011':'1/11/2011']
```

In [ ]:
```
ts.truncate(after='1/9/2011')
```

In [ ]:
```
XXXXX

dates = pd.date_range('1/1/2000', periods=100, freq='W-WED')
long_df = pd.DataFrame(np.random.randn(100, 4),
  index=dates,  columns=['Colorado', 'Texas', 'New York', 'Ohio'])
long_df.ix['5-2001']
```

In [ ]:
```
#Time Series with Duplicate Indices
dates = pd.DatetimeIndex(['1/1/2000', '1/2/2000', '1/2/2000', '1/2/2000',
  '1/3/2000'])
dup_ts = pd.Series(np.arange(5), index=dates)
print(dup_ts.index.is_unique)
print( dup_ts['1/3/2000'])
print(dup_ts['1/2/2000'])
```

In [ ]:
```
#to aggregate the data having non-unique timestamps
grouped = dup_ts.groupby(level=0)
grouped.mean()
```

In [ ]:
```
#Date Ranges, Frequencies, and Shifting

#Generic time series in pandas are assumed to be irregular;
#that is, they have no fixed frequency
#it's often desirable to work relative to a fixed frequency,
#such as daily, monthly, or every 15 minutes
print(ts)
#converting it to fixed daily frequency
print(ts.resample('D'))
```

In [ ]:
```
#Generating Date Ranges

index = pd.date_range('4/1/2012', '6/1/2012')
index
```

In [ ]:
```
pd.date_range(start='4/1/2012', periods=20)
```

In [ ]:
```
pd.date_range(end='6/1/2012', periods=20)
```

In [ ]:
```
#Frequencies and Date Offsets
#Frequencies in pandas are composed of a base frequency and a multiplier.
#Base frequencies are typically referred to by a string alias, like 'M' for monthly or
#For each base frequency, there is an object defined generally referred to as a date of

from pandas.tseries.offsets import Hour, Minute
hour = Hour()
hour

four_hours = Hour(4)
four_hours
```

In [ ]:
```
# explicitly create one of these objects,instead using a string alias like 'H' or '4H'
instead using a string alias like 'H' or '4H'.
    pd.date_range('1/1/2000', '1/3/2000 23:59', freq='4h')
```

In [ ]:
```
#pass frequency strings like '2h30min' which will effectively be parsed
#to the same expression:
x=pd.date_range('1/1/2000', periods=10, freq='1h30min')
print(x, type(x))
```

In [16]:
```
#Week of month dates
ds = pd.date_range('1/1/2012', '9/1/2012', freq='WOM-3FRI')
list(ds)
```

```
ds = pd.date_range('1/1/2012', '9/1/2012', freq='WOM-2SAT')
print(ds)
```

```
DatetimeIndex(['2012-01-14', '2012-02-11', '2012-03-10', '2012-04-14',
               '2012-05-12', '2012-06-09', '2012-07-14', '2012-08-11'],
              dtype='datetime64[ns]', freq='WOM-2SAT')
```

In [17]:

```
#Shifting (Leading and Lagging) Data
#Lag shifts a column down by a certain number.
#Lead shifts a column up by a certain number.

ts = pd.Series(np.random.randn(4),
 index=pd.date_range('1/1/2000', periods=4, freq='M'))
ts
ts.shift(2)
ts.shift(-2)
```

Out[17]:

```
2000-01-31   -1.062809
2000-02-29    2.641161
2000-03-31         NaN
2000-04-30         NaN
Freq: M, dtype: float64
```

In [ ]:

```
#example
df = pd.DataFrame({'day': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                   'sales': [18, 10, 14, 13, 19, 24, 25, 29, 15, 18]})

df['sales_previous_day'] = df['sales'].shift(1)
df['sales_previous_day2'] = df['sales'].shift(2)
df
```

In [ ]: