```
Out[11]: DatetimeIndex(['2015-07-04', '2015-07-05', '2015-07-06', '2015-07-07',
                        '2015-07-08', '2015-07-09', '2015-07-10', '2015-07-11',
                        '2015-07-12', '2015-07-13', '2015-07-14', '2015-07-15'],
                       dtype='datetime64[ns]', freq=None)
```

In the next section, we will take a closer look at manipulating time series data with the tools provided by Pandas.

Pandas Time Series: Indexing by Time

Where the Pandas time series tools really become useful is when you begin to *index data by timestamps*. For example, we can construct a `Series` object that has time-indexed data:

```
In[12]: index = pd.DatetimeIndex(['2014-07-04', '2014-08-04',
                                  '2015-07-04', '2015-08-04'])
        data = pd.Series([0, 1, 2, 3], index=index)
        data

Out[12]: 2014-07-04    0
         2014-08-04    1
         2015-07-04    2
         2015-08-04    3
         dtype: int64
```

Now that we have this data in a `Series`, we can make use of any of the `Series` indexing patterns we discussed in previous sections, passing values that can be coerced into dates:

```
In[13]: data['2014-07-04':'2015-07-04']

Out[13]: 2014-07-04    0
         2014-08-04    1
         2015-07-04    2
         dtype: int64
```

There are additional special date-only indexing operations, such as passing a year to obtain a slice of all data from that year:

```
In[14]: data['2015']

Out[14]: 2015-07-04    2
         2015-08-04    3
         dtype: int64
```

Later, we will see additional examples of the convenience of dates-as-indices. But first, let's take a closer look at the available time series data structures.

Pandas Time Series Data Structures

This section will introduce the fundamental Pandas data structures for working with time series data:

- For *time stamps*, Pandas provides the `Timestamp` type. As mentioned before, it is essentially a replacement for Python's native `datetime`, but is based on the more efficient `numpy.datetime64` data type. The associated index structure is `DatetimeIndex`.
- For *time periods*, Pandas provides the `Period` type. This encodes a fixed-frequency interval based on `numpy.datetime64`. The associated index structure is `PeriodIndex`.
- For *time deltas* or *durations*, Pandas provides the `Timedelta` type. `Timedelta` is a more efficient replacement for Python's native `datetime.timedelta` type, and is based on `numpy.timedelta64`. The associated index structure is `TimedeltaIndex`.

The most fundamental of these date/time objects are the `Timestamp` and `DatetimeIndex` objects. While these class objects can be invoked directly, it is more common to use the `pd.to_datetime()` function, which can parse a wide variety of formats. Passing a single date to `pd.to_datetime()` yields a `Timestamp`; passing a series of dates by default yields a `DatetimeIndex`:

```
In[15]: dates = pd.to_datetime([datetime(2015, 7, 3), '4th of July, 2015',
                                '2015-Jul-6', '07-07-2015', '20150708'])
        dates

Out[15]: DatetimeIndex(['2015-07-03', '2015-07-04', '2015-07-06', '2015-07-07',
                        '2015-07-08'],
                       dtype='datetime64[ns]', freq=None)
```

Any `DatetimeIndex` can be converted to a `PeriodIndex` with the `to_period()` function with the addition of a frequency code; here we'll use `'D'` to indicate daily frequency:

```
In[16]: dates.to_period('D')

Out[16]: PeriodIndex(['2015-07-03', '2015-07-04', '2015-07-06', '2015-07-07',
                      '2015-07-08'],
                     dtype='int64', freq='D')
```

A `TimedeltaIndex` is created, for example, when one date is subtracted from another:

```
In[17]: dates - dates[0]

Out[17]:
TimedeltaIndex(['0 days', '1 days', '3 days', '4 days', '5 days'],
               dtype='timedelta64[ns]', freq=None)
```

Regular sequences: pd.date_range()

To make the creation of regular date sequences more convenient, Pandas offers a few functions for this purpose: `pd.date_range()` for timestamps, `pd.period_range()` for periods, and `pd.timedelta_range()` for time deltas. We've seen that Python's

`range()` and NumPy's `np.arange()` turn a startpoint, endpoint, and optional stepsize into a sequence. Similarly, `pd.date_range()` accepts a start date, an end date, and an optional frequency code to create a regular sequence of dates. By default, the frequency is one day:

```
In[18]: pd.date_range('2015-07-03', '2015-07-10')
```

```
Out[18]: DatetimeIndex(['2015-07-03', '2015-07-04', '2015-07-05', '2015-07-06',
                         '2015-07-07', '2015-07-08', '2015-07-09', '2015-07-10'],
                        dtype='datetime64[ns]', freq='D')
```

Alternatively, the date range can be specified not with a start- and endpoint, but with a startpoint and a number of periods:

```
In[19]: pd.date_range('2015-07-03', periods=8)
```

```
Out[19]: DatetimeIndex(['2015-07-03', '2015-07-04', '2015-07-05', '2015-07-06',
                         '2015-07-07', '2015-07-08', '2015-07-09', '2015-07-10'],
                        dtype='datetime64[ns]', freq='D')
```

You can modify the spacing by altering the `freq` argument, which defaults to `D`. For example, here we will construct a range of hourly timestamps:

```
In[20]: pd.date_range('2015-07-03', periods=8, freq='H')
```

```
Out[20]: DatetimeIndex(['2015-07-03 00:00:00', '2015-07-03 01:00:00',
                         '2015-07-03 02:00:00', '2015-07-03 03:00:00',
                         '2015-07-03 04:00:00', '2015-07-03 05:00:00',
                         '2015-07-03 06:00:00', '2015-07-03 07:00:00'],
                        dtype='datetime64[ns]', freq='H')
```

To create regular sequences of period or time delta values, the very similar `pd.period_range()` and `pd.timedelta_range()` functions are useful. Here are some monthly periods:

```
In[21]: pd.period_range('2015-07', periods=8, freq='M')
```

```
Out[21]:
PeriodIndex(['2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12',
             '2016-01', '2016-02'],
            dtype='int64', freq='M')
```

And a sequence of durations increasing by an hour:

```
In[22]: pd.timedelta_range(0, periods=10, freq='H')
```

```
Out[22]:
TimedeltaIndex(['00:00:00', '01:00:00', '02:00:00', '03:00:00', '04:00:00',
                '05:00:00', '06:00:00', '07:00:00', '08:00:00', '09:00:00'],
               dtype='timedelta64[ns]', freq='H')
```

All of these require an understanding of Pandas frequency codes, which we'll summarize in the next section.

Frequencies and Offsets

Fundamental to these Pandas time series tools is the concept of a frequency or date offset. Just as we saw the `D` (day) and `H` (hour) codes previously, we can use such codes to specify any desired frequency spacing. Table 3-7 summarizes the main codes available.

Table 3-7. Listing of Pandas frequency codes

Code	Description	Code	Description
`D`	Calendar day	`B`	Business day
`W`	Weekly		
`M`	Month end	`BM`	Business month end
`Q`	Quarter end	`BQ`	Business quarter end
`A`	Year end	`BA`	Business year end
`H`	Hours	`BH`	Business hours
`T`	Minutes		
`S`	Seconds		
`L`	Milliseonds		
`U`	Microseconds		
`N`	Nanoseconds		

The monthly, quarterly, and annual frequencies are all marked at the end of the specified period. Adding an `S` suffix to any of these marks it instead at the beginning (Table 3-8).

Table 3-8. Listing of start-indexed frequency codes

Code	Description
`MS`	Month start
`BMS`	Business month start
`QS`	Quarter start
`BQS`	Business quarter start
`AS`	Year start
`BAS`	Business year start