In [1]:
```
%matplotlib inline
from matplotlib import style
style.use('fivethirtyeight')
import matplotlib.pyplot as plt
```

In [2]:
```
import numpy as np
import pandas as pd
```

In [3]:
```
import datetime as dt
```

Reflect Tables into SQLAlchemy ORM

In [4]:
```
# Python SQL toolkit and Object Relational Mapper
import sqlalchemy
from sqlalchemy.ext.automap import automap_base
from sqlalchemy.orm import Session
from sqlalchemy import create_engine, func
```

In [7]:
```
engine = create_engine("sqlite:///hawaii.sqlite")
```

In [8]:
```
# reflect an existing database into a new model
Base = automap_base()
# reflect the tables
Base.prepare(engine, reflect=True)
```

In [9]:
```
# We can view all of the classes that automap found
Base.classes.keys()
```

Out[9]:
```
['measurement', 'station']
```

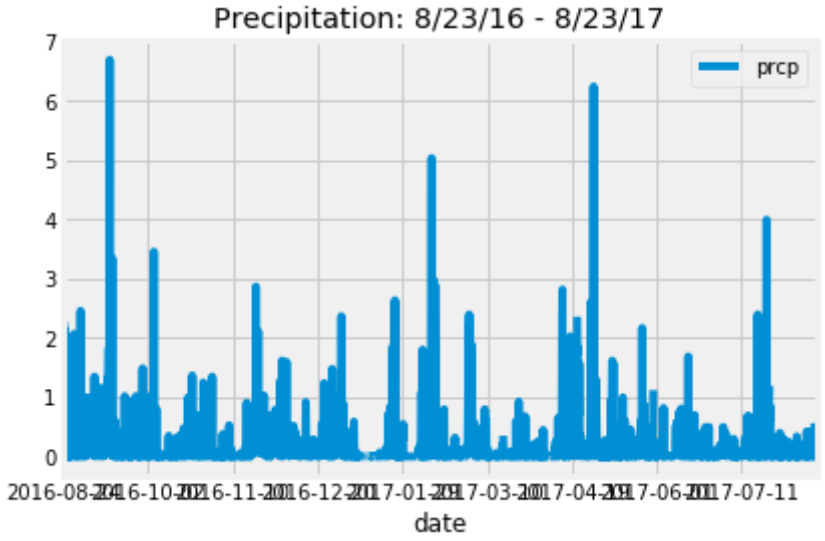
In [10]:
```
# Save references to each table
Measurement = Base.classes.measurement
Station = Base.classes.station
```

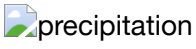
In [11]:
```
# Create our session (link) from Python to the DB
session = Session(engine)
```

Exploratory Climate Analysis

In [12]:
```
latestDate = (session.query(Measurement.date)
                     .order_by(Measurement.date.desc())
                     .first())


latestDate = list(np.ravel(latestDate))[0]

latestDate = dt.datetime.strptime(latestDate, '%Y-%m-%d')


latestYear = int(dt.datetime.strftime(latestDate, '%Y'))
latestMonth = int(dt.datetime.strftime(latestDate, '%m'))
latestDay = int(dt.datetime.strftime(latestDate, '%d'))


yearBefore = dt.date(latestYear, latestMonth, latestDay) - dt.timedelta(day


rainData = (session.query(Measurement.date, Measurement.prcp)
                   .filter(Measurement.date > yearBefore)
                   .order_by(Measurement.date)
                   .all())


rainTable = pd.DataFrame(rainData)
rainTable = rainTable.set_index('date')


rainTable = rainTable.sort_index(ascending=True)


rainTable.plot(title="Precipitation: 8/23/16 - 8/23/17")
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1144a1e48>





In [13]:
```
rainTable.describe()
```

Out[13]:

	prcp
count	2015.000000
mean	0.176462
std	0.460288
min	0.000000
25%	0.000000
50%	0.020000
75%	0.130000
max	6.700000

In [14]:
```
stationsCount = session.query(Station).count()
print(f"Station Count: {stationsCount}")
```

```
Station Count: 9
```

In [15]:
```
stationCounts = (session.query(Measurement.station, func.count(Measurement.
                        .group_by(Measurement.station)
                        .order_by(func.count(Measurement.station).desc())
                        .all())
stationCounts
```

Out[15]:
```
[('USC00519281', 2772),
 ('USC00519397', 2724),
 ('USC00513117', 2709),
 ('USC00519523', 2669),
 ('USC00516128', 2612),
 ('USC00514830', 2202),
 ('USC00511918', 1979),
 ('USC00517948', 1372),
 ('USC00518838', 511)]
```

In [14]:
```
# Using the station id from the previous query, calculate the lowest temper
# highest temperature recorded, and average temperature of the most active
```

Out[14]:
```
[(54.0, 85.0, 71.66378066378067)]
```

In [1]:
```
# Choose the station with the highest number of temperature observations.
# Query the last 12 months of temperature observation data for this station
```

In [16]:
```
# This function called `calc_temps` will accept start date and end date in
# and return the minimum, average, and maximum temperatures for that range
def calc_temps(start_date, end_date):
    """TMIN, TAVG, and TMAX for a list of dates.

    Args:
        start_date (string): A date string in the format %Y-%m-%d
        end_date (string): A date string in the format %Y-%m-%d

    Returns:
        TMIN, TAVE, and TMAX
    """

    return session.query(func.min(Measurement.tobs), func.avg(Measurement.t
        filter(Measurement.date >= start_date).filter(Measurement.date <= e

# function usage example
print(calc_temps('2012-02-28', '2012-03-05'))
```

```
[(62.0, 69.57142857142857, 74.0)]
```

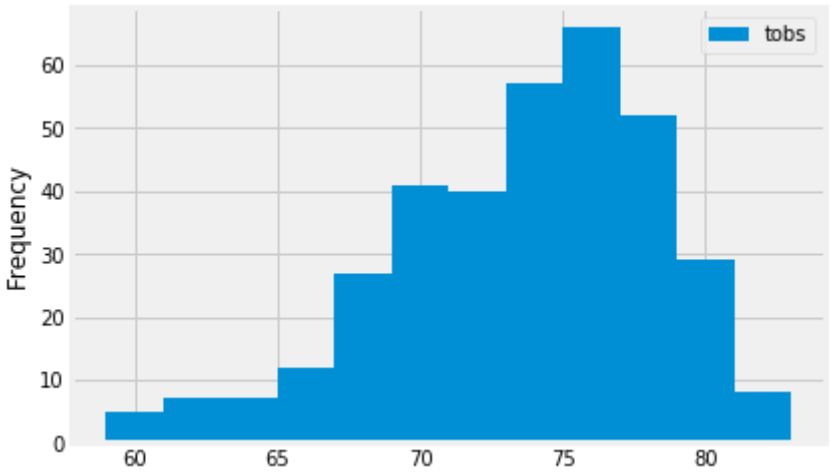
In [17]:

```
stationID = stationCounts[0][0]


stationName = (session.query(Station.name)
                      .filter_by(station = stationID))
stationName = stationName[0][0]
print(f"The most active station is {stationID}: {stationName}.")


highestTemp = (session.query(Measurement.tobs)
                      .filter(Measurement.station == stationID)
                      .order_by(Measurement.tobs.desc())
                      .first())
highestTemp = highestTemp[0]
print(f"The highest temperature recorded there is {highestTemp} degrees Far


lowestTemp = (session.query(Measurement.tobs)
                     .filter(Measurement.station == stationID)
                     .order_by(Measurement.tobs.asc())
                     .first())
lowestTemp = lowestTemp[0]
print(f"The lowest temperature recorded there is {lowestTemp} degrees Faren


avgTemp = (session.query(func.avg(Measurement.tobs))
                  .filter(Measurement.station == stationID))
avgTemp = '{0:.3}'.format(avgTemp[0][0])
print(f"The average temperature recorded there is {avgTemp} degrees Farenhe
```

```
The most active station is USC00519281: WAIHEE 837.5, HI US.
The highest temperature recorded there is 85.0 degrees Farenheit.
The lowest temperature recorded there is 54.0 degrees Farenheit.
The average temperature recorded there is 71.7 degrees Farenheit.
```

In [18]:
```
tempData = (session.query(Measurement.date, Measurement.tobs)
                   .filter(Measurement.date > yearBefore)
                   .filter(Measurement.station == stationID)
                   .order_by(Measurement.date)
                   .all())


tempTable = pd.DataFrame(tempData)
tempTable = tempTable.set_index('date')


tempTable = tempTable.sort_index(ascending=True)


tempTable.plot(kind='hist', bins=12)
```

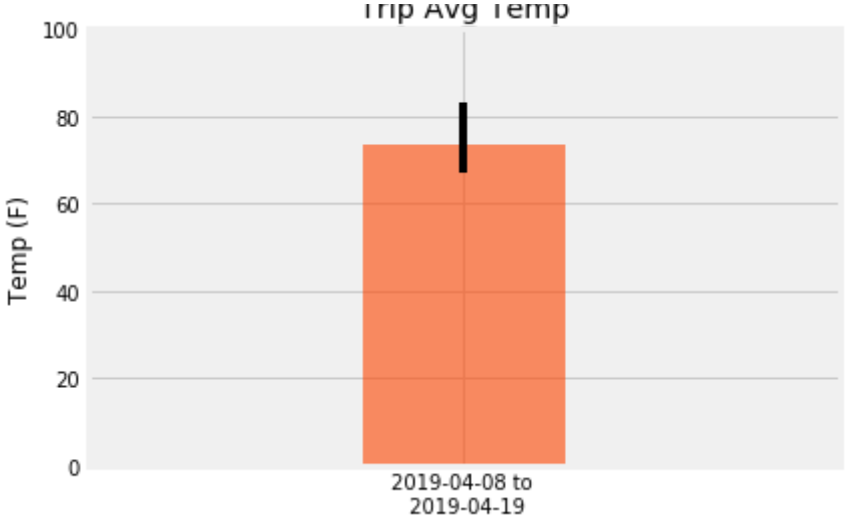
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x11486e978>



In [21]:
```
def calc_temps(start_date, end_date):
    """TMIN, TAVG, and TMAX for a list of dates.

    Args:
        start_date (string): A date string in the format %Y-%m-%d
        end_date (string): A date string in the format %Y-%m-%d

    Returns:
        TMIN, TAVE, and TMAX
    """

    return session.query(func.min(Measurement.tobs), func.avg(Measurement.t
        filter(Measurement.date >= start_date).filter(Measurement.date <= e


print(calc_temps('2012-02-28', '2012-03-05'))

trip = '2019-04-08 to \n 2019-04-19'
tripStartDate = '2017-04-08'
tripEndDate = '2017-04-19'

tripTemps = calc_temps(tripStartDate, tripEndDate)

tripTemps

minTripTemp = tripTemps[0][0]
avgTripTemp = tripTemps[0][1]
maxTripTemp = tripTemps[0][2]



minError = avgTripTemp - minTripTemp
maxError = maxTripTemp - avgTripTemp

errorBars = np.array([[minError], [maxError]])

plt.bar(trip, avgTripTemp, yerr=errorBars, color = 'orangered', alpha = .6)
plt.ylim(0, 100)
plt.xlim(-1.5, 1.5)
plt.title('Trip Avg Temp')
plt.ylabel('Temp (F)')
```

```
[(62.0, 69.57142857142857, 74.0)]
```

Out[21]:
```
Text(0, 0.5, 'Temp (F)')
```



Optional Challenge Assignment

In [20]:
```
# Create a query that will calculate the daily normals
# (i.e. the averages for tmin, tmax, and tavg for all historic data matchin

def daily_normals(date):
    """Daily Normals.

    Args:
        date (str): A date string in the format '%m-%d'

    Returns:
        A list of tuples containing the daily normals, tmin, tavg, and tmax

    """

    sel = [func.min(Measurement.tobs), func.avg(Measurement.tobs), func.max
    return session.query(*sel).filter(func.strftime("%m-%d", Measurement.da

daily_normals("01-01")
```

Out[20]:
```
[(62.0, 69.15384615384616, 77.0)]
```

In [21]:
```
# calculate the daily normals for your trip
# push each tuple of calculations into a list called `normals`

# Set the start and end date of the trip

# Use the start and end date to create a range of dates

# Stip off the year and save a list of %m-%d strings

# Loop through the list of %m-%d strings and calculate the normals for each
```

Out[21]:
```
[(62.0, 69.15384615384616, 77.0),
 (60.0, 69.39622641509433, 77.0),
 (62.0, 68.9090909090909, 77.0),
 (58.0, 70.0, 76.0),
 (56.0, 67.96428571428571, 76.0),
 (61.0, 68.96491228070175, 76.0),
 (57.0, 68.54385964912281, 76.0)]
```

In [22]:
```
# Load the previous query results into a Pandas DataFrame and add the `trip
```

Out[22]:

date	tmin	tavg	tmax
2018-01-01	62.0	69.153846	77.0
2018-01-02	60.0	69.396226	77.0
2018-01-03	62.0	68.909091	77.0
2018-01-04	58.0	70.000000	76.0
2018-01-05	56.0	67.964286	76.0

In [23]:
```
# Plot the daily normals as an area plot with `stacked=False`
```

```
<IPython.core.display.Javascript object>
```