Synoptic Data API Cheat Sheet

There are three required sections in calling the Synoptic Data API
https://api.synopticdata.com/v2/stations/latest?&token={your_token}&stid=KDEN
(1) API URL Root (2) API Service (3) API Parameters

API Services

/latest - Returns the most recent observation from a station or set of stations
/timeseries - Returns data for a station or set of stations based on a time span
/precip - Returns derived precipitation totals or intervals for a requested time period
/metadata - Returns metadata (information about stations) for a station or set of stations

API Required Parameters

API token - Unique token associated with your account for authenticating API requests
Station, network, or geospatial argument - Define a station or pattern of stations to retrieve

API Common Parameters

Variable - Prescribe what vars you want returned (grabs all variables if not set)
Units - Choose between metric or english
Geospatial arguments - Set bbox, radius, country, state, county, NWS or Fire zones
Timing arguments - Set start and end, or within for /timeseries, or set last for /latest
QC -

Common issues

Exceeding individual API request limit of 100,000 SUs (Station Unit = #times * #vars * #stations)
Requesting an end time in the future

Receiving the data

Web browser - Install JSONVue extension for greater readability
Terminal - 'get' utility
Python

```
with req.urlopen(url) as url:
        data = json.loads(url.read().decode())
```

Other useful tools

Getting started - https://developers.synopticdata.com/mesonet/v2/getting-started/
Coding examples - https://developers.synopticdata.com/mesonet/v2/examples/
More services and details at https://developers.synopticdata.com
API Query Builder - https://developers.synopticdata.com/mesonet/explorer/
Download tool (manually download station data) - https://download.synopticdata.com/
Metadata Explorer Tool (station, network, metadata viewer) - https://explore.synopticdata.com/

A couple useful functions for making api requests and parsing the responses in Python

1. Making an API request

```
def make_api_request(url, api_args):
    """Build the api request from the url and api_args, make the request, and parse
    the json return to a dictionary

    Parameters:
        url: str, url of the api endpoint
        api_args: dict, api arguments

    Returns:
        output: dict, api request response
    """
    # Append the api arguments on to the url
    for argument, value in api_args.items():
        url = url + '&' + argument + '=' + value

    # Make the api request
    print(f"API request: {url}")
    with req.urlopen(url) as response:
        body = response.read()

    # parse the json response.
    try:
        output = json.loads(body)
    except:
        decoded_body = body.decode('latin1')
        output = json.loads(decoded_body)

    return output
```

2. Parsing the response into a Pandas dataframe

```
def return_station_df(data, service):
    """Build pandas dataframes for data and metadata using json response from
        requests to Time Series, Nearest, and Latest services

    Parameters:
        data: dict, json response from API request
        date_format: str, requested date format
        service: str, Synoptic web service requested

    Returns:
        data_df: pandas DataFrame, data return from all station
        meta_df: pandas DataFrame, station metadata
    """
    dattim_format = '%Y-%m-%d %H:%M'
    meta_list = []
```

```
    # We iterate over the list of stations
    for i in range(len(data)):
        # Append station metadata to a grand list that we'll convert to a df
        stid = data[i]['STID']
        mnet_id = data[i]['MNET_ID']
        try:
            lon = float(data[i]['LONGITUDE'])
        except TypeError:
            lon = None
        try:
            lat = float(data[i]['LATITUDE'])
        except TypeError:
            lat = None
        try:
            elev = float(data[i]['ELEVATION'])
        except TypeError:
            elev = None
        meta_list.append([stid, mnet_id, lon, lat, elev])

        # Create a multi-index object to attach to the data df
        data_out = data[i]['OBSERVATIONS'].copy()
        if service == 'timeseries':
            datetime = pd.to_datetime(data_out['date_time'], format=(dattim_format))
            del data_out['date_time']
            multi_index = pd.MultiIndex.from_product([[stid], datetime],
                                                     names=["stid", "dattim"])
        else:
            datetime = pd.to_datetime(data_out[list(data_out.keys())[0]]['date_time'],
format=(dattim_format))
            for key in data_out:
                data_out.update({key: data_out[key]['value']})
            multi_index = pd.MultiIndex.from_arrays([[stid], [datetime]],
                                                    names=["stid", "dattim"])

        # Build the data df, concatenating as needed
        if i == 0:
            data_df = pd.DataFrame(data_out, index=multi_index)
        else:
            data_df = pd.concat([data_df, pd.DataFrame(data_out, index=multi_index)], axis=0)

    #Build metadata dataframe from list
    meta_df = pd.DataFrame(meta_list, columns=["stid", "mnet_id", "lon", "lat", "elev"])
    meta_df.set_index('stid', inplace=True)

    # Sort the resulting data dataframe by time
    data_df.sort_index(inplace=True)

    return data_df, meta_df
```