Name: Galapia, Xander Sam E.

Section: CPE22S3

```
import requests

def make_request(endpoint, payload = None):
  """
  Make a request to a specific endpoint on the weather API
  passing headers and optional payload.

  Parameters:
    - endpoint: The endpoint of the API you want to make a GET request to.
    - payload, A dictionary of data to pass along with the request.

  Returns:
    Response object.
  """
  return requests.get(f'https://www.ncdc.noaa.gov/cdo-web/api/v2/{endpoint}',\
                      headers={'token' : 'UktGEIDTENEtDdmCZdsafPPIgLoHXTpX'},
      params=payload
  )

  response = make_request('datasets', {'stardate': '2018-10-01'})
  response.status_code
```

```
import requests

def make_request(endpoint, payload = None):
  """
  Make a request to a specific endpoint on the weather API
  passing headers and optional payload.

  Parameters:
    - endpoint: The endpoint of the API you want to make a GET request to.
    - payload, A dictionary of data to pass along with the request.

  Returns:
    Response object.
  """
  return requests.get(f'https://www.ncdc.noaa.gov/cdo-web/api/v2/{endpoint}',
                      headers={'token' : 'UktGEIDTENEtDdmCZdsafPPIgLoHXTpX'},params=payload
  )



response = make_request('datasets', {'startdate': '2024-13-03'})
response.status_code
```

```
200
```

```
response.json().keys()
```

```
dict_keys(['metadata', 'results'])
```

```
response.json()['metadata']
```

```
{'resultset': {'offset': 1, 'count': 11, 'limit': 25}}
```

```
response.json()['results'][0].keys()
```

```
dict_keys(['uid', 'mindate', 'maxdate', 'name', 'datacoverage', 'id'])
```

Parse the Result

```
[(data['id'], data['name']) for data in response.json()['results']]
```

```
[('GHCND', 'Daily Summaries'),
 ('GSOM', 'Global Summary of the Month'),
 ('GSOY', 'Global Summary of the Year'),
```

```
      ('NEXRAD2', 'Weather Radar (Level II)'),
      ('NEXRAD3', 'Weather Radar (Level III)'),
      ('NORMAL_ANN', 'Normals Annual/Seasonal'),
      ('NORMAL_DLY', 'Normals Daily'),
      ('NORMAL_HLY', 'Normals Hourly'),
      ('NORMAL_MLY', 'Normals Monthly'),
      ('PRECIP_15', 'Precipitation 15 Minute'),
      ('PRECIP_HLY', 'Precipitation Hourly')]


#Get data catageory ID
response = make_request(
    'datacategories',
    payload = {
        'datasetid' : 'GHCND'
    }
)
response.status_code
```

```
     200
```

```
response.json()['results']
```

```
     [{'name': 'Evaporation', 'id': 'EVAP'},
      {'name': 'Land', 'id': 'LAND'},
      {'name': 'Precipitation', 'id': 'PRCP'},
      {'name': 'Sky cover & clouds', 'id': 'SKY'},
      {'name': 'Sunshine', 'id': 'SUN'},
      {'name': 'Air Temperature', 'id': 'TEMP'},
      {'name': 'Water', 'id': 'WATER'},
      {'name': 'Wind', 'id': 'WIND'},
      {'name': 'Weather Type', 'id': 'WXTYPE'}]
```

```
#get data type id
response = make_request(
    'datatypes',
    payload = {
        'datacategoryid' : 'TEMP',
        'limit' : 100
    }
)
response.status_code
```

```
     200
```

```
[(datatype['id'], datatype['name']) for datatype in response.json()['results']][-5:]
```

```
     [('MNTM', 'Monthly mean temperature'),
      ('TAVG', 'Average Temperature.'),
      ('TMAX', 'Maximum temperature'),
      ('TMIN', 'Minimum temperature'),
      ('TOBS', 'Temperature at the time of observation')]
```

```
#get location category id
response = make_request(
    'locationcategories',
    {
        'datasetid' : 'GHCND'
    }
)
response.status_code
```

```
     200
```

```
import pprint
pprint.pprint(response.json())
```

```
     {'metadata': {'resultset': {'count': 12, 'limit': 25, 'offset': 1}},
      'results': [{'id': 'CITY', 'name': 'City'},
                  {'id': 'CLIM_DIV', 'name': 'Climate Division'},
                  {'id': 'CLIM_REG', 'name': 'Climate Region'},
                  {'id': 'CNTRY', 'name': 'Country'},
                  {'id': 'CNTY', 'name': 'County'},
                  {'id': 'HYD_ACC', 'name': 'Hydrologic Accounting Unit'},
                  {'id': 'HYD_CAT', 'name': 'Hydrologic Cataloging Unit'},
                  {'id': 'HYD_REG', 'name': 'Hydrologic Region'},
                  {'id': 'HYD_SUB', 'name': 'Hydrologic Subregion'},
```

```
                {'id': 'ST', 'name': 'State'},
                {'id': 'US_TERR', 'name': 'US Territory'},
                {'id': 'ZIP', 'name': 'Zip Code'}]}


def get_item(name,what,endpoint, start = 1, end = None):
  """
  Grab the JSON payload for a given field by name using binary search.

  Parameters:
   - name: The item to look for.
   - what: Dictioanry specifying what the item in 'name' is.
   - endpoint: Where to look for the item.
   - start: The position of the cities. Used to find the midpoint, but
      like 'start' this is not something we need to worry about.

  Returns:
    Dictionary of the information for the item if found otherwise
    an empty dictionary.
  """
  #FIND the midpoint which we use to cut the data in half each time
  mid = (start + (end if end else 1)) // 2

  #LOWERCASE the name so this is not case-sensitive
  name = name.lower()

  #DEFINE the payload we will send with each request
  payload = {
      'datasetid' : 'GHCND',
      'sortfield' : 'name',
      'offset' : mid, #We'll change the offset each time
      'limit' : 1 #We only want one value check
  }

  #MAKE our request adding any additional filter parameters from 'what'
  response = make_request(endpoint, {**payload, **what})

  if response.ok:
    #IF response is ok, grab the end index from the response metadata teh first
    #time through
    end = end if end else response.json()['metadata']['resultset']['count']

    #GRAB the lowestcase version of the current name
    current_name = response.json()['results'][0]['name'].lower()

    #IF what we are searching for is in the current name, we have found our item
    if name in current_name:
      return response.json()['results'][0] #RETURN THE FOUND ITEM
    else:
      if start >= end:
        #IF our start index is greater than or equal to our end, we couldn't find it
        return{}
      elif name < current_name:
        #OUR name comes before the current name in the alphabet, so we search further to the left
        return get_item(name,what,endpoint,start, mid-1)
      elif name > current_name:
        #our name comes after the current name in the alphabet, so we search further to the right
        return get_item(name, what, endpoint, mid + 1, end)
  else:
    #RESPONSE wasn't okay , use code to determine why
    print(f'Response not OK, status: {response.status_code}')

def get_location(name):
  """
  GRAB the JSON payload for the location by name using binary search

  Parameters:
    - name: The city to look for.

  Returns:
    Dictionary of the information for the city if found otherwise
    an empty dictionary"""

  return get_item(name, {'locationcategoryid' : 'CITY'}, 'locations')
```

```
#GET NYC ID
nyc = get_location('New York')
nyc
```

```
{'mindate': '1869-01-01',
 'maxdate': '2024-03-11',
 'name': 'New York, NY US',
 'datacoverage': 1,
 'id': 'CITY:US360019'}
```

```
central_park = get_item('NY City Central Park', {'locationid' : nyc['id']}, 'stations')
central_park
```

```
{'elevation': 42.7,
 'mindate': '1869-01-01',
 'maxdate': '2024-03-10',
 'latitude': 40.77898,
 'name': 'NY CITY CENTRAL PARK, NY US',
 'datacoverage': 1,
 'id': 'GHCND:USW00094728',
 'elevationUnit': 'METERS',
 'longitude': -73.96925}
```

```
#GET NYC daily summaries data
response = make_request(
    'data',
    {
        'datasetid' : 'GHCND',
        'stationid' : central_park['id'],
        'locationid' : nyc['id'],
        'startdate' : '2018-10-01',
        'enddate' : '2018-10-31',
        'datatypeid' : ['TMIN', 'TMAX', 'TOB'],
        'units' : 'metric',
        'limit' : 1000
    }
)
response.status_code
```

```
200
```

```
import pandas as pd

df = pd.DataFrame(response.json()['results'])
df.head()
```

	date	datatype	station	attributes	value
0	2018-10-01T00:00:00	TMAX	GHCND:USW00094728	,,W,2400	24.4
1	2018-10-01T00:00:00	TMIN	GHCND:USW00094728	,,W,2400	17.2
2	2018-10-02T00:00:00	TMAX	GHCND:USW00094728	,,W,2400	25.0
3	2018-10-02T00:00:00	TMIN	GHCND:USW00094728	,,W,2400	18.3
4	2018-10-03T00:00:00	TMAX	GHCND:USW00094728	,,W,2400	23.3

Next steps: View recommended plots

```
df.datatype.unique()
```

```
array(['TMAX', 'TMIN'], dtype=object)
```

```
if get_item(
    'NY City Central Park', {'locationid' : nyc['id'], 'datatypeid' : 'TOBS'}, 'stations'
):
  print('Found')
```

```
Found
```

```
laguardia = get_item(
    'LaGuardia' , {'locationid' : nyc['id']}, 'stations'
)
laguardia
```

```
{'elevation': 3,
 'mindate': '1939-10-07',
 'maxdate': '2024-03-11',
 'latitude': 40.77945,
 'name': 'LAGUARDIA AIRPORT, NY US',
 'datacoverage': 1,
 'id': 'GHCND:USW00014732',
```

```
#GET NYC Daily Summaries Data

response = make_request(
    'data',
    {
        'datasetid' : 'GHCND',
        'stationid' : laguardia['id'],
        'locationid' : nyc['id'],
        'startdate' : '2018-10-01',
        'enddate' : '2018-10-31',
        'datatypeid' : ['TMIN', 'TMAX', 'TAVG'], #Temperature at time of observation, min, max, average
        'units' : 'metric',
        'limit' : 1000
    }
)
response.status_code
```

```
200
```

```
df = pd.DataFrame(response.json()['results'])
df.head()
```

	date	datatype	station	attributes	value
0	2018-10-01T00:00:00	TAVG	GHCND:USW00014732	H,,S,	21.2
1	2018-10-01T00:00:00	TMAX	GHCND:USW00014732	,,W,2400	25.6
2	2018-10-01T00:00:00	TMIN	GHCND:USW00014732	,,W,2400	18.3
3	2018-10-02T00:00:00	TAVG	GHCND:USW00014732	H,,S,	22.7
4	2018-10-02T00:00:00	TMAX	GHCND:USW00014732	,,W,2400	26.1

Next steps: View recommended plots

```
df.datatype.value_counts()
```

```
TAVG    31
TMAX    31
TMIN    31
Name: datatype, dtype: int64
```

```
df.to_csv('/content/nyc_temperatures.csv', index=False)
```