```
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': 'KFwvUbvvdsWskOidlBysYFNxyWtdtVcV'
    },
    params=payload
  )



response = make_request('datasets', {'startdate':'2018-10-01'})
response.status_code # if 200, request is success
```

```
200
```

```
# response object can be accessed with .json()
response.json().keys() # keys are the main identifiers (?)
```

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

```
response.json()['metadata'] # metadata are information about the request
```

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

```
response.json()['results'][0].keys() # results are the actual data results we requested
```

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

```
response.json()['results'] # prints the actual results for clearer picture
```

```
  'maxdate': '2024-02-01',
  'name': 'Global Summary of the Month',
  'datacoverage': 1,
  'id': 'GSOM'},
 {'uid': 'gov.noaa.ncdc:C00947',
  'mindate': '1763-01-01',
  'maxdate': '2024-01-01',
  'name': 'Global Summary of the Year',
  'datacoverage': 1,
  'id': 'GSOY'},
 {'uid': 'gov.noaa.ncdc:C00345',
  'mindate': '1991-06-05',
  'maxdate': '2024-03-12',
  'name': 'Weather Radar (Level II)',
  'datacoverage': 0.95,
  'id': 'NEXRAD2'},
 {'uid': 'gov.noaa.ncdc:C00708',
  'mindate': '1994-05-20',
  'maxdate': '2024-03-10',
  'name': 'Weather Radar (Level III)',
  'datacoverage': 0.95,
  'id': 'NEXRAD3'},
 {'uid': 'gov.noaa.ncdc:C00821',
```

```
       'name': 'Normals Daily',
       'datacoverage': 1,
       'id': 'NORMAL_DLY'},
      {'uid': 'gov.noaa.ncdc:C00824',
       'mindate': '2010-01-01',
       'maxdate': '2010-12-31',
       'name': 'Normals Hourly',
       'datacoverage': 1,
       'id': 'NORMAL_HLY'},
      {'uid': 'gov.noaa.ncdc:C00822',
       'mindate': '2010-01-01',
       'maxdate': '2010-12-01',
       'name': 'Normals Monthly',
       'datacoverage': 1,
       'id': 'NORMAL_MLY'},
      {'uid': 'gov.noaa.ncdc:C00505',
       'mindate': '1970-05-12',
       'maxdate': '2014-01-01',
       'name': 'Precipitation 15 Minute',
       'datacoverage': 0.25,
       'id': 'PRECIP_15'},
      {'uid': 'gov.noaa.ncdc:C00313',
       'mindate': '1900-01-01',
       'maxdate': '2014-01-01',
       'name': 'Precipitation Hourly',
       'datacoverage': 1,
       'id': 'PRECIP_HLY'}]
```

```
[(data['id'], data['name']) for data in response.json()['results']] # gets the id and name of data
```

```
     [('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 category id
response = make_request( # changes the content of response variable
  'datacategories',
  payload={
    'datasetid' : 'GHCND' # gets the GHCND, daily summaries
  }
)
response.status_code # returns 200, request success
```

```
     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 # get 100 data points
  }
)
response.status_code # returns 200, request success
```

```
     200
```

Check the documentation: https://www.ncdc.noaa.gov/cdo-web/webservices/v2,

and look at the endpoints for the general categories

```
[(datatype['id'], datatype['name']) for datatype in response.json()['results']][-5:] # look at the last 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' # recall as daily summaries
  }
)
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: Dictionary specifying what the item in `name` is.
  - endpoint: Where to look for the item.
  - start: The position to start at. We don't need to touch this, but the
  function will manipulate this with recursion.
  - end: The last 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 will change the offset each time
    'limit' : 1 # we only want one value back
  }
  # 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 the first time through
    end = end if end else response.json()['metadata']['resultset']['count']
    # grab the lowercase 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 ok, 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'}


# gets the central park station at NY
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', # daily summaries
    'stationid' : central_park['id'], # from central park station
    'locationid' : nyc['id'], # at ny
    'startdate' : '2018-10-01', # gets from oct 1, 2018
    'enddate' : '2018-10-31', # to oct 31, 2018
    'datatypeid' : ['TMIN', 'TMAX', 'TOBS'], # temperature at time of observation, min, and max
    'units' : 'metric',
    'limit' : 1000
  }
)
response.status_code
```

```
     200
```

```
import pandas as pd
df = pd.DataFrame(response.json()['results']) # transforms to a pandas dataframe
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()
# we want to get the temp at time of observation (TOBS)
# however, only TMAX and TMIN are available
```

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

```
if get_item(
  'NY City Central Park', {'locationid' : nyc['id'], 'datatypeid': 'TOBS'}, 'stations'
):
  print('Found!')
# it found a value which has TOBS,
# which means that maybe the TOBS is at a different date
```

```
     Found!
```

```
laguardia = get_item( # changed station to laguardia airport
  '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',
      'elevationUnit': 'METERS',
      'longitude': -73.88027}
```

```
# 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, and max
    'units' : 'metric',
    'limit' : 1000
  }
)
response.status_code
```

```
200
```

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

	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