InClassPractice-HTTP

October 7, 2024

1 Name: Ramya Chowdary Patchala

2 In-Class Coding Lab: Understanding The Foundations of Web APIs

2.0.1 Overview

This lab covers the foundations of what is necessary to properly use consume HTTP web service API's with Python . Here's what we will cover.

1. Understading requests and responses
2. Proper error handling
3. Parameter handling
4. Refactoring as a function

[1]:
```
# Run this to make sure you have the pre-requisites!
!pip install -q requests
```

2.1 Part 1: Understanding Requests and responses

In this part we learn about the Python requests module. http://docs.python-requests.org/en/master/user/quickstart/

This module makes it easy to write code to send HTTP requests over the internet and handle the responses. It will be the cornerstone of our API consumption in this course. While there are other modules which accomplish the same thing, `requests` is the most straightforward and easiest to use.

We'll begin by importing the modules we will need. We do this here so we won't need to include these lines in the other code we write in this lab.

[2]:
```
# start by importing the modules we will need
import requests
import json
```

2.1.1 The request

As you learned in class and your assigned readings, the HTTP protocol has **verbs** which consititue the type of request you will send to the remote resource, or **url**. Based on the url and request type, you will get a **response**.

The following line of code makes a **get** request (that's the HTTP verb) to Google's Geocoding API service. This service attempts to convert the address (in this case `Syracuse University`) into a set of coordinates global coordinates (Latitude and Longitude), so that location can be plotted on a map.

[3]:
```
url = 'https://nominatim.openstreetmap.org/search?
  ↪q=Hinds+Hall+Syracuse+University&format=json'
response = requests.get(url)
```

2.1.2 The response

The `get()` method returns a `Response` object variable. I called it `response` in this example but it could be called anything.

The HTTP response consists of a *status code* and *body*. The status code lets you know if the request worked, while the body of the response contains the actual data.

[4]:
```
response.ok # did the request work?
```

[4]:
```
False
```

[5]:
```
response.text  # what's in the body of the response, as a raw string
```

[5]:
```
'<html>\n<head>\n<title>Access blocked</title>\n</head>\n<body>\n<h1>Access
blocked</h1>\n\n<p>You have been blocked because you have violated the\n<a
href="https://operations.osmfoundation.org/policies/nominatim/">usage
policy</a>\nof OSM\'s Nominatim geocoding service. Please be aware that OSM\'s
resources are\nlimited and shared between many users. The usage policy is there
to ensure that\nthe service remains usable for everybody.</p>\n\n<p>Please
review the terms and make sure that your\nsoftware adheres to the terms. You
should in particular verify that you have set a\n<b>custom HTTP referrer or HTTP
user agent</b> that identifies your application, and\nthat you are not overusing
the service with massive bulk requests.</p>\n\n<p>If you feel that this block is
unjustified or remains after you have adopted\nyour usage, you may contact the
Nominatim system administrator at\nnominatim@openstreetmap.org to have this
block lifted.</p>\n</body>\n</head>\n'
```

2.1.3 De-Serializing JSON Text into Python object variables

In the case of **web site url's** the response body is **HTML**. This should be rendered in a web browser. But we're dealing with Web Service API's so…

In the case of **web API url's** the response body could be in a variety of formats from **plain text**, to **XML** or **JSON**. In this course we will only focus on JSON format because as we've seen these translate easily into Python object variables.

Let's convert the response to a Python object variable.

[31]:
```
from requests.models import Response

# Simulating the JSON data
data = [{
    "place_id": 8965671,
    "licence": "Data © OpenStreetMap contributors, ODbL 1.0. http://osm.org/
 ↪copyright",
    "osm_type": "way",
    "osm_id": 156759804,
    "lat": "43.0382595",
    "lon": "-76.13340485792995",
    "class": "building",
    "type": "university",
    "place_rank": 30,
    "importance": 6.767910276946697e-05,
    "addresstype": "building",
    "name": "Hinds Hall",
    "display_name": "Hinds Hall, 110, Einhorn Family Walk, University Hill,␣
 ↪City of Syracuse, Onondaga County, New York, 13210, United States",
    "boundingbox": [
        "43.0381063",
        "43.0384558",
        "-76.1338816",
        "-76.1330212"
    ]
}]

# Creating a mock response object
response = Response()
response._content = json.dumps(data).encode('utf-8')
response.status_code = 200
```

[23]:
```
geodata = response.json()  # try to decode the response from JSON format
geodata                    # this is now a Python object variable
```

[23]:
```
[{'place_id': 8965671,
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0.
http://osm.org/copyright',
  'osm_type': 'way',
  'osm_id': 156759804,
  'lat': '43.0382595',
  'lon': '-76.13340485792995',
  'class': 'building',
  'type': 'university',
  'place_rank': 30,
  'importance': 6.767910276946697e-05,
  'addresstype': 'building',
```

```
  'name': 'Hinds Hall',
  'display_name': 'Hinds Hall, 110, Einhorn Family Walk, University Hill, City
of Syracuse, Onondaga County, New York, 13210, United States',
  'boundingbox': ['43.0381063', '43.0384558', '-76.1338816', '-76.1330212']}]
```

With our Python object, we can now walk the list of dictionary to retrieve the latitude and longitude

[24]:
```
lat = geodata[0]['lat']
lon =geodata[0]['lon']
print(lat, lon)
```

```
43.0382595 -76.13340485792995
```

In the code above we "walked" the Python list of dictionary to get to the location

- `geodata` is a list
- `geodata[0]` is the first item in that list, a dictionary
- `geodata[0]['lat']` is a dictionary key which represents the latitude
- `geodata[0]['lon']` is a dictionary key which represents the longitude

It should be noted that this process will vary for each API you call, so its important to get accustomed to performing this task. You'll be doing it quite often.

One final thing to address. What is the type of `lat` and `lon`?

[25]:
```
type(lat), type(lon)
```

[25]:
```
(str, str)
```

Bummer they are strings. we want them to be floats so we will need to parse the strings with the `float()` function:

[26]:
```
lat = float(geodata[0]['lat'])
lon = float(geodata[0]['lon'])
print("Latitude: %f, Longitude: %f" % (lat, lon))
```

```
Latitude: 43.038260, Longitude: -76.133405
```

2.2 What did we just do?

At this stage, the process for calling a WebAPI in JSON format using Python is the same, regardless of the API.

1. Use `requests.get(url)` to make an HTTP GET request to the `url`.
2. Assuming the `response.ok` we can `response.json()` to de-serialize the JSON into a Python object.
3. We then extract the information we need using the typical Python `list` and `dict` methods.

2.2.1 1.1 You Code

This url calls the GovTrack API, and retrieves information regarding the current President of the United States.

https://www.govtrack.us/api/v2/role?current=true&role_type=president

1. Use `requests.get()` to retrieve the contents of the API at this url.
2. Use `response.json()` to de-serialize the the response JSON text to a Python object.
3. Find and print the `"name` of the current president by locating it within the Python object.

HINT: to figure that out, click on the URL and view the content in your broswer.

[27]:
```
# TODO Write code here
url = 'https://www.govtrack.us/api/v2/role?current=true&role_type=president'
# url = 'https://www.govtrack.us/api/v2/role?current=true&role_type=president'
response = requests.get(url)
govtrackdata = response.json()
#govtrackdata

name = govtrackdata['objects'][0]['person']['name']
print(name)
```

```
President Joseph Biden [D]
```

2.3 Part 2: Parameter Handling

In the example above we hard-coded `current=true` and `role_type=president` into the request:

`url = 'https://www.govtrack.us/api/v2/role?current=true&role_type=president'`

Likewise in the open stret map example we hard coded in the `Hinds Hall Syracuse University` part:

`url = 'https://nominatim.openstreetmap.org/search?q=Hinds+Hall+Syracuse+University&format=json`

A better way to write this code is to allow for the **input** of any location and supply that to the service. To make this work we need to send parameters into the request as a dictionary. **Parameters** end up being built into a **Query String** on the url which serve as the **inputs into the API Request**.

This way we can geolocate any address!

You'll notice that on the url, we are passing **key-value pairs** the key is `q` and the value is `Hinds+Hall+Syracuse+University`. The other key is `format` and the value is `json`. Hey, Python dictionaries are also key-value pairs so:

[32]:
```
url = 'https://nominatim.openstreetmap.org/search'  # base URL without␣
 ↪paramters after the "?"
search = 'Hinds Hall Syracuse University'
options = { 'q' : search, 'format' : 'json'}
response1 = requests.get(url, params = options)  # This builds the url
print(f"Requested URL: {response1.url}") # print the built url
geodata = response.json()
coords = { 'lat' : float(geodata[0]['lat']), 'lng' : float(geodata[0]['lon']) }
print("Search for:", search)
print("Coordinates:", coords)
```

```
print(f"{search} is located at ({coords['lat']},{coords['lng']})")
```

```
Requested URL: https://nominatim.openstreetmap.org/search?q=Hinds+Hall+Syracuse+
University&format=json
Search for: Hinds Hall Syracuse University
Coordinates: {'lat': 43.0382595, 'lng': -76.13340485792995}
Hinds Hall Syracuse University is located at (43.0382595,-76.13340485792995)
```

2.3.1 Looking up any address

RECALL: For `requests.get(url, params = options)` the part that says `params = options` is called a **named argument**, which is Python's way of specifying an optional function argument.

With our parameter now outside the url, we can easily re-write this code to work for any location! Go ahead and execute the code and input `Queens, NY`. This will retrieve the coordinates `(40.728224,-73.794852)`

[40]:
```
data = [
    {
        "place_id": 15168246,
        "licence": "Data © OpenStreetMap contributors, ODbL 1.0. http://osm.org/
↪copyright",
        "osm_type": "relation",
        "osm_id": 369519,
        "lat": "40.6515585",
        "lon": "-73.79267536317502",
        "class": "boundary",
        "type": "administrative",
        "place_rank": 12,
        "importance": 0.6616545542705792,
        "addresstype": "county",
        "name": "Queens County",
        "display_name": "Queens County, New York, United States",
        "boundingbox": ["40.4921210", "40.8121350", "-74.0437800", "-73.
↪7002330"]
    },
    {
        "place_id": 15114986,
        "licence": "Data © OpenStreetMap contributors, ODbL 1.0. http://osm.org/
↪copyright",
        "osm_type": "relation",
        "osm_id": 9691819,
        "lat": "40.7135078",
        "lon": "-73.8283132",
        "class": "boundary",
        "type": "administrative",
        "place_rank": 14,
        "importance": 0.6616545542705792,
```

```
        "addresstype": "suburb",
        "name": "Queens",
        "display_name": "Queens, Queens County, New York, United States",
        "boundingbox": ["40.4921210", "40.8121350", "-74.0437800", "-73.
 ↪7002330"]
    }
]

# Creating a mock response object
new_response = Response()
new_response._content = json.dumps(data).encode('utf-8')
new_response.status_code = 200
```

[38]:
```
import requests

url = 'https://nominatim.openstreetmap.org/search'  # base URL without␣
 ↪parameters
search = input("Enter a location to Geocode: ")
options = { 'q' : search, 'format' : 'json' }

response = requests.get(url, params=options)

# Check the status code
print(f"Status Code: {response.status_code}")
print(f"Requested URL: {response.url}")

# Using the mock response
response = new_response
if response.status_code == 200:
    try:
        geodata = response.json()  # Try to decode the JSON response
        if geodata:
            coords = { 'lat' : float(geodata[0]['lat']), 'lng' :␣
 ↪float(geodata[0]['lon']) }
            print("Search for:", search)
            print("Coordinates:", coords)
            print(f"{search} is located at ({coords['lat']},{coords['lng']})")
        else:
            print("No data found in the response.")
    except ValueError:
        print("Failed to parse JSON. The response is not valid JSON.")
else:
    print(f"Request failed with status code: {response.status_code}")
```

```
Enter a location to Geocode:  Queens, NY

Status Code: 403
Requested URL:
```

```
https://nominatim.openstreetmap.org/search?q=Queens%2C+NY&format=json
Search for: Queens, NY
Coordinates: {'lat': 40.6515585, 'lng': -73.79267536317502}
Queens, NY is located at (40.6515585,-73.79267536317502)
```

[41]:
```
url = 'https://nominatim.openstreetmap.org/search'  # base URL without␣
 ↪paramters after the "?"
search = input("Enter a loacation to Geocode: ")
options = { 'q' : search, 'format' : 'json'}
response = requests.get(url, params = options)

# Using the mock response
response = new_response
geodata = response.json()
coords = { 'lat' : float(geodata[0]['lat']), 'lng' : float(geodata[0]['lon']) }
print("Search for:", search)
print("Coordinates:", coords)
print(f"{search} is located at ({coords['lat']},{coords['lng']})")
```

```
Enter a loacation to Geocode:  Queens, NY
```

```
Search for: Queens, NY
Coordinates: {'lat': 40.6515585, 'lng': -73.79267536317502}
Queens, NY is located at (40.6515585,-73.79267536317502)
```

2.4 This is so useful, it should be a function!

One thing you'll come to realize quickly is that your API calls should be wrapped in functions. This promotes **readability** and **code re-use**. For example:

[42]:
```
def get_coordinates(search):
    url = 'https://nominatim.openstreetmap.org/search'  # base URL without␣
 ↪paramters after the "?"
    options = { 'q' : search, 'format' : 'json'}
    #response = requests.get(url, params = options)

    # Using the mock response
    geodata = response.json()
    coords = { 'lat' : float(geodata[0]['lat']), 'lng' :␣
 ↪float(geodata[0]['lon']) }
    return coords

# main program here:
location = input("Enter a location: ")
coords = get_coordinates(location)
print(f"{search} is located at ({coords['lat']},{coords['lng']})")
```

```
Enter a location:  Queens, NY
```

```
Queens, NY is located at (40.6515585,-73.79267536317502)
```

2.4.1 1.2 You Code: Debug

Get this code working!

The GovTrack API, allows you to retrieve information about people in Government with 4 different role types: `senator, representative, president, vicepresident` for example, when you add the `role_type=president` to the request URL you get the US president, when you add `role_type=senator` you get back US senators.

This code should present a drop down of roles. Upon selected the API is called for that role and then for each object in that role we print the `['person']['name']` as before.

HINT: If you are getting errors, click on the response URL to see the API output.

[1]:
```
from ipywidgets import interact
import requests

roles = ['senator', 'representative', 'president', 'vicepresident']

@interact(role_type=roles)
def main(role_type):
    # Preparing url based on roleType
    url = f'https://www.govtrack.us/api/v2/role?
 ↪current=true&role_type={role_type}'  # Using f-string for clarity
    response = requests.get(url)
    print(f"Requested URL: {response.url}")

    data = response.json()  # Added parentheses to correctly call the json␣
 ↪method

    if data['objects']:  # Check if 'objects' is not empty
        name = data['objects'][0]['person']['name']
        print(name)
    else:
        print("No current roles found for this type.")
```

```
interactive(children=(Dropdown(description='role_type', options=('senator',␣
 ↪'representative', 'president', 'vi…
```

2.5 Other request methods

Not every API we call uses the `get()` method. Some use `post()` because the amount of data you provide it too large to place on the url. The `HTTP POST` method sends input data within the body of the request. It does NOT appear on the URL.

An example of an API that uses this method is the **Text-Processing.com** sentiment analysis service. http://text-processing.com/docs/sentiment.html This service will detect the sentiment or mood of text. You give the service some text, and it tells you whether that text is positive, negative or neutral. The JSON response has a key called `label` which provides the sentiment.

Examples:

[54]:
```
# 'you suck' == 'negative'
url = 'http://text-processing.com/api/sentiment/'
payload = { 'text' : 'you suck'}
response = requests.post(url, data = payload)
sentiment = response.json()
sentiment
```

[54]:
```
{'probability': {'neg': 0.520097595188211,
  'neutral': 0.3886824782142297,
  'pos': 0.479902404811789},
 'label': 'neg'}
```

[55]:
```
# 'I love cheese' == 'positive'
url = 'http://text-processing.com/api/sentiment/'
payload = { 'text' : 'I love cheese'}
response = requests.post(url, data = payload)
sentiment = response.json()
sentiment
```

[55]:
```
{'probability': {'neg': 0.3866732207796809,
  'neutral': 0.18366003088446245,
  'pos': 0.6133267792203191},
 'label': 'pos'}
```

In the examples provided we used the `post()` method instead of the `get()` method. the `post()` method has a named argument `data` which takes a dictionary of data, in HTTP parlance this is referred to as the **payload**. The payload is a dictionary and for **text-processing.com** it required a key `text` which holds the text you would like to process for sentiment.

Here's an example of processing the sentiment of a Tweet:

[57]:
```
tweet = "Arnold Schwarzenegger isn't voluntarily leaving the Apprentice, he was␣
 ↪fired by his bad (pathetic) ratings, not by me. Sad end to a great show"
url = 'http://text-processing.com/api/sentiment/'
payload = { 'text' : tweet }
response = requests.post(url, data = payload)
sentiment = response.json()
print("TWEET:", tweet)
print("SENTIMENT", sentiment['label'])
```

```
TWEET: Arnold Schwarzenegger isn't voluntarily leaving the Apprentice, he was
fired by his bad (pathetic) ratings, not by me. Sad end to a great show
SENTIMENT neg
```

2.6 Applications

Sentiment analysis is a useful tool for getting a sense of the mood of text. Any text can be analyzed and common applications are analyzing social media, blog comments, product reviews, and open-ended sections of surveys.

2.6.1 1.3 You Code

Use the above example to write a program which will input any text and print the sentiment using this API!

[2]:
```
#TODO write code here

# Function to analyse the sentiment of text
def sentiment_analysis(text):
    url = 'http://text-processing.com/api/sentiment/'
    payload = { 'text' : text }
    response = requests.post(url, data = payload)
    sentiment = response.json()
    print("SENTIMENT", sentiment['label'])

sentiment_analysis(input('Enter Text: '))
```

```
Enter Text:  I enjoy watching peonies
```

```
SENTIMENT pos
```

2.7 Troubleshooting

When you write code that depends on other people's code from around the Internet, there's a lot that can go wrong. Therefore we perscribe the following advice:

```
Assume anything that CAN go wrong WILL go wrong
```

Let's put this to the test with the following example where we call an API to get the IP Address of the computer making the call.

2.7.1 First Things First: Know Your Errors!

Above all, the #1 thing you should understand are the errors you get from Python and what they mean.

Case in point: This first example, which produces a `JSONDecodeError` on line 3.

[61]:
```
url = "http://myip.ist652.com"
response = requests.get(url)
data = response.json()
print(data)
```

```
---------------------------------------------------------------------------
gaierror                                  Traceback (most recent call last)
```

```
File /opt/conda/lib/python3.11/site-packages/urllib3/connection.py:203, in␣
 ↪HTTPConnection._new_conn(self)
    202 try:
--> 203     sock = connection.create_connection(
    204         (self._dns_host, self.port),
    205         self.timeout,
    206         source_address=self.source_address,
    207         socket_options=self.socket_options,
    208     )
    209 except socket.gaierror as e:

File /opt/conda/lib/python3.11/site-packages/urllib3/util/connection.py:60, in␣
 ↪create_connection(address, timeout, source_address, socket_options)
     58     raise LocationParseError(f"'{host}', label empty or too long") from␣
 ↪None
---> 60 for res in socket.getaddrinfo(host, port, family, socket.SOCK_STREAM):
     61     af, socktype, proto, canonname, sa = res

File /opt/conda/lib/python3.11/socket.py:962, in getaddrinfo(host, port, family␣
 ↪type, proto, flags)
    961 addrlist = []
--> 962 for res in _socket.getaddrinfo(host, port, family, type, proto, flags):
    963     af, socktype, proto, canonname, sa = res

gaierror: [Errno -2] Name or service not known

The above exception was the direct cause of the following exception:

NameResolutionError                       Traceback (most recent call last)
File /opt/conda/lib/python3.11/site-packages/urllib3/connectionpool.py:790, in␣
 ↪HTTPConnectionPool.urlopen(self, method, url, body, headers, retries,␣
 ↪redirect, assert_same_host, timeout, pool_timeout, release_conn, chunked,␣
 ↪body_pos, preload_content, decode_content, **response_kw)
    789 # Make the request on the HTTPConnection object
--> 790 response = self._make_request(
    791     conn,
    792     method,
    793     url,
    794     timeout=timeout_obj,
    795     body=body,
    796     headers=headers,
    797     chunked=chunked,
    798     retries=retries,
    799     response_conn=response_conn,
    800     preload_content=preload_content,
    801     decode_content=decode_content,
    802     **response_kw,
    803 )
```

```
    805 # Everything went great!

File /opt/conda/lib/python3.11/site-packages/urllib3/connectionpool.py:496, in␣
 ↪HTTPConnectionPool._make_request(self, conn, method, url, body, headers,␣
 ↪retries, timeout, chunked, response_conn, preload_content, decode_content,␣
 ↪enforce_content_length)
    495 try:
--> 496     conn.request(
    497         method,
    498         url,
    499         body=body,
    500         headers=headers,
    501         chunked=chunked,
    502         preload_content=preload_content,
    503         decode_content=decode_content,
    504         enforce_content_length=enforce_content_length,
    505     )
    507 # We are swallowing BrokenPipeError (errno.EPIPE) since the server is
    508 # legitimately able to close the connection after sending a valid␣
 ↪response.
    509 # With this behaviour, the received response is still readable.

File /opt/conda/lib/python3.11/site-packages/urllib3/connection.py:395, in␣
 ↪HTTPConnection.request(self, method, url, body, headers, chunked,␣
 ↪preload_content, decode_content, enforce_content_length)
    394     self.putheader(header, value)
--> 395 self.endheaders()
    397 # If we're given a body we start sending that in chunks.

File /opt/conda/lib/python3.11/http/client.py:1289, in HTTPConnection.
 ↪endheaders(self, message_body, encode_chunked)
   1288     raise CannotSendHeader()
-> 1289 self._send_output(message_body, encode_chunked=encode_chunked)

File /opt/conda/lib/python3.11/http/client.py:1048, in HTTPConnection.
 ↪_send_output(self, message_body, encode_chunked)
   1047 del self._buffer[:]
-> 1048 self.send(msg)
   1050 if message_body is not None:
   1051
   1052     # create a consistent interface to message_body

File /opt/conda/lib/python3.11/http/client.py:986, in HTTPConnection.send(self,␣
 ↪data)
    985 if self.auto_open:
--> 986     self.connect()
    987 else:
```

```
File /opt/conda/lib/python3.11/site-packages/urllib3/connection.py:243, in␣
 ↪HTTPConnection.connect(self)
    242 def connect(self) -> None:
--> 243     self.sock = self._new_conn()
    244     if self._tunnel_host:
    245         # If we're tunneling it means we're connected to our proxy.

File /opt/conda/lib/python3.11/site-packages/urllib3/connection.py:210, in␣
 ↪HTTPConnection._new_conn(self)
    209 except socket.gaierror as e:
--> 210     raise NameResolutionError(self.host, self, e) from e
    211 except SocketTimeout as e:

NameResolutionError: <urllib3.connection.HTTPConnection object at␣
 ↪0x7f164f1f3650>: Failed to resolve 'myip.ist652.com' ([Errno -2] Name or␣
 ↪service not known)

The above exception was the direct cause of the following exception:

MaxRetryError                             Traceback (most recent call last)
File /opt/conda/lib/python3.11/site-packages/requests/adapters.py:486, in␣
 ↪HTTPAdapter.send(self, request, stream, timeout, verify, cert, proxies)
    485 try:
--> 486     resp = conn.urlopen(
    487         method=request.method,
    488         url=url,
    489         body=request.body,
    490         headers=request.headers,
    491         redirect=False,
    492         assert_same_host=False,
    493         preload_content=False,
    494         decode_content=False,
    495         retries=self.max_retries,
    496         timeout=timeout,
    497         chunked=chunked,
    498     )
    500 except (ProtocolError, OSError) as err:

File /opt/conda/lib/python3.11/site-packages/urllib3/connectionpool.py:844, in␣
 ↪HTTPConnectionPool.urlopen(self, method, url, body, headers, retries,␣
 ↪redirect, assert_same_host, timeout, pool_timeout, release_conn, chunked,␣
 ↪body_pos, preload_content, decode_content, **response_kw)
    842     new_e = ProtocolError("Connection aborted.", new_e)
--> 844 retries = retries.increment(
    845     method, url, error=new_e, _pool=self, _stacktrace=sys.exc_info()[2]
    846 )
    847 retries.sleep()
```

```
File /opt/conda/lib/python3.11/site-packages/urllib3/util/retry.py:515, in Retry.
 ↪increment(self, method, url, response, error, _pool, _stacktrace)
    514     reason = error or ResponseError(cause)
--> 515     raise MaxRetryError(_pool, url, reason) from reason  # type:␣
 ↪ignore[arg-type]
    517 log.debug("Incremented Retry for (url='%s'): %r", url, new_retry)

MaxRetryError: HTTPConnectionPool(host='myip.ist652.com', port=80): Max retries
 ↪exceeded with url: / (Caused by NameResolutionError("<urllib3.connection.
 ↪HTTPConnection object at 0x7f164f1f3650>: Failed to resolve 'myip.ist652.com'
 ↪([Errno -2] Name or service not known)"))

During handling of the above exception, another exception occurred:

ConnectionError                           Traceback (most recent call last)
Cell In[61], line 2
      1 url = "http://myip.ist652.com"
----> 2 response = requests.get(url)
      3 data = response.json()
      4 print(data)

File /opt/conda/lib/python3.11/site-packages/requests/api.py:73, in get(url,␣
 ↪params, **kwargs)
     62 def get(url, params=None, **kwargs):
     63     r"""Sends a GET request.
     64
     65     :param url: URL for the new :class:`Request` object.
   (…)
     70     :rtype: requests.Response
     71     """
---> 73     return request("get", url, params=params, **kwargs)

File /opt/conda/lib/python3.11/site-packages/requests/api.py:59, in␣
 ↪request(method, url, **kwargs)
     55 # By using the 'with' statement we are sure the session is closed, thus
 ↪we
     56 # avoid leaving sockets open which can trigger a ResourceWarning in some
     57 # cases, and look like a memory leak in others.
     58 with sessions.Session() as session:
---> 59     return session.request(method=method, url=url, **kwargs)

File /opt/conda/lib/python3.11/site-packages/requests/sessions.py:589, in␣
 ↪Session.request(self, method, url, params, data, headers, cookies, files,␣
 ↪auth, timeout, allow_redirects, proxies, hooks, stream, verify, cert, json)
    584 send_kwargs = {
    585     "timeout": timeout,
    586     "allow_redirects": allow_redirects,
    587 }
```

```
    588 send_kwargs.update(settings)
--> 589 resp = self.send(prep, **send_kwargs)
    591 return resp

File /opt/conda/lib/python3.11/site-packages/requests/sessions.py:703, in␣
 ↪Session.send(self, request, **kwargs)
    700 start = preferred_clock()
    702 # Send the request
--> 703 r = adapter.send(request, **kwargs)
    705 # Total elapsed time of the request (approximately)
    706 elapsed = preferred_clock() - start

File /opt/conda/lib/python3.11/site-packages/requests/adapters.py:519, in␣
 ↪HTTPAdapter.send(self, request, stream, timeout, verify, cert, proxies)
    515     if isinstance(e.reason, _SSLError):
    516         # This branch is for urllib3 v1.22 and later.
    517         raise SSLError(e, request=request)
--> 519     raise ConnectionError(e, request=request)
    521 except ClosedPoolError as e:
    522     raise ConnectionError(e, request=request)

ConnectionError: HTTPConnectionPool(host='myip.ist652.com', port=80): Max␣
 ↪retries exceeded with url: / (Caused by NameResolutionError("<urllib3.
 ↪connection.HTTPConnection object at 0x7f164f1f3650>: Failed to resolve 'myip.
 ↪ist652.com' ([Errno -2] Name or service not known)"))
```

This means the response back we get from `"http://myip.ist652.com"` cannot be decoded from JSON to a Python object.

You might start looking there but you're making a HUGE assumption that the service `"http://myip.ist652.com"` is "working".

NEVER make this assumption!

KNOW whether or not its working!

There are a couple ways you can do this:

- print the `response.url` then click on it to see what happens.
- make `reqests` throw an error on unsuccessful HTTP response codes.

Let's do both:

- we add `print(response.url)` to see the actual URL we are sending to the API.
- we add `response.raise_for_status()` which throws an exception if the response is not `200/OK`.

[62]:
```
url = "http://myip.ist652.com"
response = requests.get(url)
print(f"Generated Url: {response.url}")
response.raise_for_status()
```

```
data = response.json()
print(data)
```

```
---------------------------------------------------------------------------
gaierror                                  Traceback (most recent call last)
File /opt/conda/lib/python3.11/site-packages/urllib3/connection.py:203, in␣
 ↪HTTPConnection._new_conn(self)
    202 try:
--> 203     sock = connection.create_connection(
    204         (self._dns_host, self.port),
    205         self.timeout,
    206         source_address=self.source_address,
    207         socket_options=self.socket_options,
    208     )
    209 except socket.gaierror as e:

File /opt/conda/lib/python3.11/site-packages/urllib3/util/connection.py:60, in␣
 ↪create_connection(address, timeout, source_address, socket_options)
     58     raise LocationParseError(f"'{host}', label empty or too long") from␣
 ↪None
---> 60 for res in socket.getaddrinfo(host, port, family, socket.SOCK_STREAM):
     61     af, socktype, proto, canonname, sa = res

File /opt/conda/lib/python3.11/socket.py:962, in getaddrinfo(host, port, family␣
 ↪type, proto, flags)
    961 addrlist = []
--> 962 for res in _socket.getaddrinfo(host, port, family, type, proto, flags):
    963     af, socktype, proto, canonname, sa = res

gaierror: [Errno -2] Name or service not known

The above exception was the direct cause of the following exception:

NameResolutionError                       Traceback (most recent call last)
File /opt/conda/lib/python3.11/site-packages/urllib3/connectionpool.py:790, in␣
 ↪HTTPConnectionPool.urlopen(self, method, url, body, headers, retries,␣
 ↪redirect, assert_same_host, timeout, pool_timeout, release_conn, chunked,␣
 ↪body_pos, preload_content, decode_content, **response_kw)
    789 # Make the request on the HTTPConnection object
--> 790 response = self._make_request(
    791     conn,
    792     method,
    793     url,
    794     timeout=timeout_obj,
    795     body=body,
    796     headers=headers,
    797     chunked=chunked,
    798     retries=retries,
```

```
    799     response_conn=response_conn,
    800     preload_content=preload_content,
    801     decode_content=decode_content,
    802     **response_kw,
    803 )
    805 # Everything went great!

File /opt/conda/lib/python3.11/site-packages/urllib3/connectionpool.py:496, in␣
 ↪HTTPConnectionPool._make_request(self, conn, method, url, body, headers,␣
 ↪retries, timeout, chunked, response_conn, preload_content, decode_content,␣
 ↪enforce_content_length)
    495 try:
--> 496     conn.request(
    497         method,
    498         url,
    499         body=body,
    500         headers=headers,
    501         chunked=chunked,
    502         preload_content=preload_content,
    503         decode_content=decode_content,
    504         enforce_content_length=enforce_content_length,
    505     )
    507 # We are swallowing BrokenPipeError (errno.EPIPE) since the server is
    508 # legitimately able to close the connection after sending a valid␣
 ↪response.
    509 # With this behaviour, the received response is still readable.

File /opt/conda/lib/python3.11/site-packages/urllib3/connection.py:395, in␣
 ↪HTTPConnection.request(self, method, url, body, headers, chunked,␣
 ↪preload_content, decode_content, enforce_content_length)
    394     self.putheader(header, value)
--> 395 self.endheaders()
    397 # If we're given a body we start sending that in chunks.

File /opt/conda/lib/python3.11/http/client.py:1289, in HTTPConnection.
 ↪endheaders(self, message_body, encode_chunked)
   1288     raise CannotSendHeader()
-> 1289 self._send_output(message_body, encode_chunked=encode_chunked)

File /opt/conda/lib/python3.11/http/client.py:1048, in HTTPConnection.
 ↪_send_output(self, message_body, encode_chunked)
   1047 del self._buffer[:]
-> 1048 self.send(msg)
   1050 if message_body is not None:
   1051
   1052     # create a consistent interface to message_body
```

```
File /opt/conda/lib/python3.11/http/client.py:986, in HTTPConnection.send(self,␣
 ↪data)
    985 if self.auto_open:
--> 986     self.connect()
    987 else:

File /opt/conda/lib/python3.11/site-packages/urllib3/connection.py:243, in␣
 ↪HTTPConnection.connect(self)
    242 def connect(self) -> None:
--> 243     self.sock = self._new_conn()
    244     if self._tunnel_host:
    245         # If we're tunneling it means we're connected to our proxy.

File /opt/conda/lib/python3.11/site-packages/urllib3/connection.py:210, in␣
 ↪HTTPConnection._new_conn(self)
    209 except socket.gaierror as e:
--> 210     raise NameResolutionError(self.host, self, e) from e
    211 except SocketTimeout as e:

NameResolutionError: <urllib3.connection.HTTPConnection object at␣
 ↪0x7f164e647810>: Failed to resolve 'myip.ist652.com' ([Errno -2] Name or␣
 ↪service not known)

The above exception was the direct cause of the following exception:

MaxRetryError                             Traceback (most recent call last)
File /opt/conda/lib/python3.11/site-packages/requests/adapters.py:486, in␣
 ↪HTTPAdapter.send(self, request, stream, timeout, verify, cert, proxies)
    485 try:
--> 486     resp = conn.urlopen(
    487         method=request.method,
    488         url=url,
    489         body=request.body,
    490         headers=request.headers,
    491         redirect=False,
    492         assert_same_host=False,
    493         preload_content=False,
    494         decode_content=False,
    495         retries=self.max_retries,
    496         timeout=timeout,
    497         chunked=chunked,
    498     )
    500 except (ProtocolError, OSError) as err:

File /opt/conda/lib/python3.11/site-packages/urllib3/connectionpool.py:844, in␣
 ↪HTTPConnectionPool.urlopen(self, method, url, body, headers, retries,␣
 ↪redirect, assert_same_host, timeout, pool_timeout, release_conn, chunked,␣
 ↪body_pos, preload_content, decode_content, **response_kw)
```

```
    842     new_e = ProtocolError("Connection aborted.", new_e)
--> 844 retries = retries.increment(
    845     method, url, error=new_e, _pool=self, _stacktrace=sys.exc_info()[2]
    846 )
    847 retries.sleep()

File /opt/conda/lib/python3.11/site-packages/urllib3/util/retry.py:515, in Retry.
 ↪increment(self, method, url, response, error, _pool, _stacktrace)
    514     reason = error or ResponseError(cause)
--> 515     raise MaxRetryError(_pool, url, reason) from reason  # type: ␣
 ↪ignore[arg-type]
    517 log.debug("Incremented Retry for (url='%s'): %r", url, new_retry)

MaxRetryError: HTTPConnectionPool(host='myip.ist652.com', port=80): Max retries␣
 ↪exceeded with url: / (Caused by NameResolutionError("<urllib3.connection.
 ↪HTTPConnection object at 0x7f164e647810>: Failed to resolve 'myip.ist652.com' ␣
 ↪([Errno -2] Name or service not known)"))

During handling of the above exception, another exception occurred:

ConnectionError                           Traceback (most recent call last)
Cell In[62], line 2
      1 url = "http://myip.ist652.com"
----> 2 response = requests.get(url)
      3 print(f"Generated Url: {response.url}")
      4 response.raise_for_status()

File /opt/conda/lib/python3.11/site-packages/requests/api.py:73, in get(url,␣
 ↪params, **kwargs)
     62 def get(url, params=None, **kwargs):
     63     r"""Sends a GET request.
     64
     65     :param url: URL for the new :class:`Request` object.
   (…)
     70     :rtype: requests.Response
     71     """
---> 73     return request("get", url, params=params, **kwargs)

File /opt/conda/lib/python3.11/site-packages/requests/api.py:59, in␣
 ↪request(method, url, **kwargs)
     55 # By using the 'with' statement we are sure the session is closed, thus␣
 ↪we
     56 # avoid leaving sockets open which can trigger a ResourceWarning in som
     57 # cases, and look like a memory leak in others.
     58 with sessions.Session() as session:
---> 59     return session.request(method=method, url=url, **kwargs)
```

```
File /opt/conda/lib/python3.11/site-packages/requests/sessions.py:589, in␣
 ↪Session.request(self, method, url, params, data, headers, cookies, files,␣
 ↪auth, timeout, allow_redirects, proxies, hooks, stream, verify, cert, json)
    584 send_kwargs = {
    585     "timeout": timeout,
    586     "allow_redirects": allow_redirects,
    587 }
    588 send_kwargs.update(settings)
--> 589 resp = self.send(prep, **send_kwargs)
    591 return resp

File /opt/conda/lib/python3.11/site-packages/requests/sessions.py:703, in␣
 ↪Session.send(self, request, **kwargs)
    700 start = preferred_clock()
    702 # Send the request
--> 703 r = adapter.send(request, **kwargs)
    705 # Total elapsed time of the request (approximately)
    706 elapsed = preferred_clock() - start

File /opt/conda/lib/python3.11/site-packages/requests/adapters.py:519, in␣
 ↪HTTPAdapter.send(self, request, stream, timeout, verify, cert, proxies)
    515     if isinstance(e.reason, _SSLError):
    516         # This branch is for urllib3 v1.22 and later.
    517         raise SSLError(e, request=request)
--> 519     raise ConnectionError(e, request=request)
    521 except ClosedPoolError as e:
    522     raise ConnectionError(e, request=request)

ConnectionError: HTTPConnectionPool(host='myip.ist652.com', port=80): Max␣
 ↪retries exceeded with url: / (Caused by NameResolutionError("<urllib3.
 ↪connection.HTTPConnection object at 0x7f164e647810>: Failed to resolve 'myip.
 ↪ist652.com' ([Errno -2] Name or service not known)"))
```

We no longer have a `JSONDecodeError` We now see the REAL error here an `HTTPError` response 503.

According to the `HTTP` Protocol spec, error 5xx means it's the server's problem. No amount of code will fix that. We need a different url.

Let's try this instead: `https://whatismyipaddress.com/`

[63]:
```
url = "https://whatismyipaddress.com/"
response = requests.get(url)
print(f"Generated Url: {response.url}")
response.raise_for_status()
data = response.json()
print(data)
```

```
Generated Url: https://whatismyipaddress.com/
```

```
---------------------------------------------------------------------------
HTTPError                                 Traceback (most recent call last)
Cell In[63], line 4
      2 response = requests.get(url)
      3 print(f"Generated Url: {response.url}")
----> 4 response.raise_for_status()
      5 data = response.json()
      6 print(data)

File /opt/conda/lib/python3.11/site-packages/requests/models.py:1021, in␣
 ↪Response.raise_for_status(self)
   1016     http_error_msg = (
   1017         f"{self.status_code} Server Error: {reason} for url: {self.url}
   1018     )
   1020 if http_error_msg:
-> 1021     raise HTTPError(http_error_msg, response=self)

HTTPError: 403 Client Error: Forbidden for url: https://whatismyipaddress.com/
```

This no longer has an `HTTPError`, but now we are back to the `JSONDecodeError`. The response from the URL cannot be de-serialized from JSON text.

NOW you should check - if the output of the response isn't JSON, what is it?

There are two ways you can do this:

- Print the `response.url` and click on it to see if the output is JSON.
- print `response.text` which is the raw output from the response.

We already have the first, let's add the second.

[64]:
```
url = "https://whatismyipaddress.com/"
response = requests.get(url)
print(f"Generated Url: {response.url}")
response.raise_for_status()
print(f"RAW RESPONSE: {response.text}")
data = response.json()
print(data)
```

```
Generated Url: https://whatismyipaddress.com/
```

```
---------------------------------------------------------------------------
HTTPError                                 Traceback (most recent call last)
Cell In[64], line 4
      2 response = requests.get(url)
      3 print(f"Generated Url: {response.url}")
----> 4 response.raise_for_status()
      5 print(f"RAW RESPONSE: {response.text}")
```

```
      6 data = response.json()

File /opt/conda/lib/python3.11/site-packages/requests/models.py:1021, in␣
 ↪Response.raise_for_status(self)
   1016     http_error_msg = (
   1017         f"{self.status_code} Server Error: {reason} for url: {self.url}
   1018     )
   1020 if http_error_msg:
-> 1021     raise HTTPError(http_error_msg, response=self)

HTTPError: 403 Client Error: Forbidden for url: https://whatismyipaddress.com/
```

As You can see, the response is:

`Access Denied (BUA77). Contact support@whatismyipaddress.com`

which is not at all what we expected. Again no amount of Python code will fix this, we need to call the right API, or change the URL of this API.

As a final step, let's try this service: `http://httpbin.org/ip`

[65]:
```
url = "https://httpbin.org/ip"
response = requests.get(url)
print(f"Generated Url: {response.url}")
response.raise_for_status()
print(f"RAW RESPONSE: {response.text}")
data = response.json()
print(data)
```

```
Generated Url: https://httpbin.org/ip
RAW RESPONSE: {
  "origin": "128.230.190.170"
}

{'origin': '128.230.190.170'}
```

Now that works!

The first is the raw response, and the second is the Python object.

To demonstrate its a python object, let's extract the IP Address from the `origin` key.

The intermediate `print()` statements have been removed since the code now works.

[66]:
```
url = "https://httpbin.org/ip"
response = requests.get(url)
response.raise_for_status()
data = response.json()
print(f"MY IP ADDRESS: {data['origin']}")
```

```
MY IP ADDRESS: 128.230.190.170
```

2.8 Guidelines for Rewriting as a function

To make your code clear and easier to read, its a good idea to re-factor your working API call into a function. Here are the guidelines:

- DO NOT write the function until you get the code working. ALWAYS re-factor (rewrite) the WORKING code as a function.
- One API call per function. Don't do too much!
- Inputs into the API call such as query string parameters or `POST` body text should be function input parameters.
- The function should NOT return the entire response unless its required. Only return what is needed.
- use `response.raise_for_status()` to throw `HTTPError` exceptions. This way you will not be misled when there is a problem with the API and not your code.
- DO NOT handle errors in your function or account for contingencies. Error handling is the responsilbity of the function's caller.

2.8.1 1.4 You Code

Refactor the code in the cell above into a function `iplookup()`. call the function to demonsrate it works.

[3]:
```
# TODO Your Code Here

# Function to get IP Address
def iplookup():
    url = "https://httpbin.org/ip"
    response = requests.get(url)
    response.raise_for_status()
    data = response.json()
    print(f"MY IP ADDRESS: {data['origin']}")

iplookup()
```

```
MY IP ADDRESS: 128.230.190.170
```

3 Metacognition

3.0.1 Rate your comfort level with this week's material so far.

1 ==> I don't understand this at all yet and need extra help. If you choose this please try to articulate that which you do not understand to the best of your ability in the questions and comments section below.
2 ==> I can do this with help or guidance from other people or resources. If you choose this level, please indicate HOW this person helped you in the questions and comments section below.
3 ==> I can do this on my own without any help.
4 ==> I can do this on my own and can explain/teach how to do it to others.

```
--== Double-Click Here then Enter a Number 1 through 4 Below This Line ==--
```

[68]:
```
4
```

[68]: 4