DAY 1
About HTTP and API's

REST APIs:

- API stands for Application Programming Interface
- An API allows you to search for something, and feeds back results from whatever service you are requesting it.
- An API lets a developer create programs that talk to one another, and exchange data.
- A ***REST*** API stands for "Representational State Transfer". Which means that this API adheres to some rules.
- Representational deals with how resources get manipulated and represented. Data usually gets represented in XML / JSON.
- An URL sent to the API Service, is known as a **request**, while the data sent back is known as the **response**.
- REST is Resource Based (Deals with Things rather than Actions and Nouns rather than Verbs) , SOAP is vice-versa.
- 6 Constraints :
    - **Uniform Interface**
        - Defines the interface between Client and Server
        - Simplifies and separates the architecture
        - HTTP verbs, URIs and HTTP response
    - **Stateless**
        - Server contains no client
        - Requests have enough context for the server to process the message
        - All states on client only
        - Some services like OAuth are Stateful but built in a REST fashion.
    - **Client-Server**
        - Disconnected System
        - Separation of Concerns
        - Uniform Interface is the link.
    - **Cacheable**
        - Data that comes back from a service can be cached.
        - **Implicit Caching -** Client Caches
        - **Explicit Caching -** Server caches
        - Negotiated Caching - Interaction between C and S occurs
    - **Layered System**
        - Client can't assume direct connection to server
        - Combo of Client-Server and Cacheable
        - Improves scalability
    - **Code on Demand (Optional)**
        - Server transfers logic to client to reduce load on itself
        - Client executes the logic

- Applets, JS

What makes a request:

1. Endpoint and Paths:
    a. The URL you request from
    b. E.g., https://api.github.com or https://api.twitter.com
    c. The path after the endpoint determines which resource you are requesting for.
    d. https://api.twitter.com/tag/something refers to the something tag in the API resource codebase
    e. To understand all the available paths, one must thoroughly read the documentation of the API.
    f. To get a list of repos of a user through GitHub's API, one would use `/users/:username/repos` where ":" refers to a variable.
    g. **Query parameters** are present in APIs, although they aren't a part of REST architecture.
        i. Gives us the option to modify the request with key-value pairs.
        ii. Always begin with a question mark "?".
        iii. Seperated with an ampersand "&".
        iv. E.g,https://api.github.com/users/viswalahiri/repos?sort=created&direction=asc
    h. Testing Endpoints:
        i. Endpoints can be tested through different methods
            1. Javascript - Fetch API
            2. jQuery - Ajax method
            3. Ruby = Net::HTTP class
            4. Python - Requests
        ii. cURL
            1. APIs can be tested using cURL, and their documentations are generally written with reference to cURL.
            2. cURL can be installed from https://curl.haxx.se/windows/
            3. `curl https://api.github.com/users/zellwk/repos`
            4. If query parameters exists, then prepend ? and = with "\"
            `curl https://api.github.com/users/zellwk/repos`
    i. JSON
        i. JavaScript Object Notation is a common format for sending and requesting data through a REST API.
        ii. Each property and value must be wrapped around double quotation marks.
        iii. `{ "property1": "value1", "property2": "value2" }`

2. Method:

a. The method describes the type of request you send to the server.
b. There are 5 different types of requests that are used to perform CRUD (Create ,Read, Update, Delete.
    i. GET
        1. The default request
        2. Equivalent to READ
        3. Get a resource from the server
        4. Server looks for the data, and sends it back
    ii. POST
        1. Equivalent to CREATE
        2. Used to create a new resource on server
        3. Creates new entry in DB, and tells whether successful
    iii. PUT and PATCH
        1. Equivalent to UPDATE
        2. Updates an entry in the server DB and tells whether successful
        3. PUT is complete update, whereas PATCH is a partial update
    iv. DELETE
        1. Equivalent to DELETE
        2. Deletes entry from DB and tells whether successful.
c. Examples
    i. GET : Get a list of repos of GitHub
    ii. POST : Create a new GitHub Repo
d. Methods in cURL
    i. We can set the request method in cURL using -X or --request.
    ii. `curl -X POST https://api.github.com/user/repos`

3. Headers:
    a. Used to provide/exchange information between client and server.
    b. Used for authentication, and providing content for the body.
    c. HTTP headers are **property-value-pairs**, that are separated by a colon
        i. E.g., "Content-Type: application/json". Missing the opening ".
    d. Headers through cURL
        i. Send HTTP Headers through cURL with -H or --headers.
            1. `curl -H "Content-Type: application/json" https://api.github.com`
        ii. View headers you've sent using the -v or --verbose option as you send the request.
            1. `curl -H "Content-Type: application/json" https://api.github.com -v`
        iii. '>' refers to the request, whereas '<' refers to the response on the Command Line Console.

4. Data / Body:
    a. Contains information you want to be sent to servers, and is generally done using POST, PUT, PATCH, or DELETE.
    b. To send data over cURL, we can use -d or --data
        i. `curl -X POST <URL> -d property1=value1`
        ii. `curl -X POST <URL> -d property1=value1 -d property2=value2`
        iii. Requests can be separated into multiple lines by adding a "\".
            1. curl -X POST <URL> \
               -d property1=value1 \
                -d property2=value2

Authentication

1. Authentication is needed, as POST, PUT, PATCH, and DELETE requests alter the DB.
2. In some cases even a GET requires authentication
3. Ways of authentication
    a. Basic Authentication (With a username and password)
    b. Secret Token Authentication - oAuth
4. Performing Basic Authentication with cURL, one can use -u followed by the username and password
    a. E.g. curl -x POST -u "username:password" https://api.github.com/user/repos

HTTP Status Codes

1. Codes range from 100+ to 500+
2. The numbers follow the following rules
    a. 200+ (Request has succeeded)
    b. 300+ (Request is redirected)
    c. 400+ (Client Error)
    d. 500+ (Server Error)
3. Response status can be debugged with the verbose option -V or --verbose, or the head option -I or --head.

API Versions

1. Sometimes APIs get updated, and when they do, your application break
2. You can request for a specific version of the API in either two ways
    a. Directly in the endpoint
        i. https://api.twitter.com/1.1/account/settings.json
    b. In a request header
        i. curl https://api.github.com -H Accept:application/vnd.github.v3+json

Interacting with REST API's

1. In Python, we use the HTTP library "requests".
2. A code snippet is as follows

```
import requests
resp = requests.get('https://todolist.example.com/tasks/')
if resp.status_code != 200:
    # This means something went wrong.
    raise ApiError('GET /tasks/ {}'.format(resp.status_code))
for todo_item in resp.json():
    print('{} {}'.format(todo_item['id'], todo_item['summary']))
```

3. Json methods are to be used for correct formatting. This can be done by using the methods, json.dumps(), json.loads()