**What is an API?**

* “API allows for information or functionality to be manipulated by other programs via the internet.”
* Application Programming Interface = part of a computer program designed to be used or manipulated by another **program**.
* (As opposed to being used by another human)

**When to create an API?**

1. Large dataset.
2. Data to be accessed in real-time.
3. Data will be changed/be updated frequently.
4. Users (other programs) only access data one part at a time.
5. Advanced actions to be performed other than retrieve data, such as:
   1. Contributing
   2. Updating data
   3. Deleting data.

* If you have data to share to the world of developers, use an API.
* If data set size relatively small just provide a ‘data dump’, but can be beneficial to provide API and data dump.

**Data dump = size of data relatively small, instead provide a ‘data dump’ in form of a downloadable JSON/ XML/ CSV/ SQLite file.**

**API Terminology:**

1. **HTTP:**

* HyperText Transfer Protocol, primary means of **data communication on the web**.
* **‘Methods’** used to show data direction: [ ‘GET’, ‘POST’,....]

1. **URL (“https://programminghistorian.org/about”)**

* **Address** for a resource on the web.
* Consists of:

1. Protocol - **https://**
2. Path - **programminghistorian.org/about**
3. **JSON:**

* JavaScript Object Notation, text-based data storage format.
* Designed to be easy to read for humans **and** machines.
* Most common **format for returning data** through API.

1. **REST**

* REpresenational State Transfer, describes best practices for implementing API’s.
* API designed with some or all rules are called RESTful API.
* Can be disagreement around the principles needed, so may refer tomas web API or HTTP API.

**Endpoint = a URL that enables the API to access resources on a server, often through a RESTful API interface.**

**Using API’s**

* Case study - Sensationalism and Historical Fires

Our research area is sensationalism and the press: has newspaper coverage of major events in the US become more or less sensational over time? Narrowing the topic, we may ask whether coverage of *urban fires* has increased or decreased with government reporting on fire-related relief spending?

**Begin by collecting historical data** on newspaper coverage of fires using an API - <http://chroniclingamerica.loc.gov/about/api/>. This API does not require an authentication process.

**Our initial goal is to find *all newspaper stories in the database that use the term ‘fire’*.**

Typically any use of an API starts with its documentation, we find two critical piece of information:

1. API’s **Base URL**

**http://chroniclingamerica.loc.gov**

1. **Path** corresponding to functions you want to preform (search)

/search/pages/results/

1. If we **combine the ‘Base URL’ and the ‘path’ into one URL**, we will create a request to the API that returns all available data in the DB.

<http://chroniclingamerica.loc.gov/search/pages/results/>

This will return data on all items available at the time of writing in a *formatted HTML view*. If we want structured data (JSON) and related to fires, we need to pass **Query Parameters** (?):

1. ***format=json***- changes returning data from HTML to JSON.
2. ***proxtext=fire***- narrows returned entries to those with search term ‘fire’.

<http://chroniclingamerica.loc.gov/search/pages/results/?format=json&proxtext=fire>

* Query Parameters follow the “**?**” in the URL, and separated by “**&**”.
* Example output structure:

**"city": ["Washington"], "date": "19220611", "title": "The Washington herald. [volume]", "end\_year": 1939, "note": ["Also issued on microfilm from the Library of Congress, Photoduplication Service.", "Archived issues are available in digital format as part of the Library of Congress Chronicling America online collection.", "Nov. 19-20 both called vol 1, no. 1", "On Sunday published as: Washington times-herald, Nov. 19, 1922-Apr. 15, 1923; Washington herald times, Sept. 26, 1937-Jan. 29, 1939."], "state": ["District of Columbia"], "section\_label": "", "type": "page", "place\_of\_publication": "Washington, D.C.", "start\_year": 1906, "edition\_label": "Sunday Edition", "publisher": null, "language": ["English"], "alt\_title": ["Washington herald times", "Washington times-herald"], "lccn": "sn83045433", "country": "District of Columbia", "ocr\_eng": "Mrs. Harding with representative# of the Camp Fire Girto.", "batch": "dlc\_greyhound\_ver02", "title\_normal": "washington herald.", "url": "https://chroniclingamerica.loc.gov/lccn/sn83045433/1922-06-11/ed-1/seq-37.json", "place": ["District of Columbia--Washington"], "page": "Page 5"}, {"sequence": 32, "county": [null], "edition": null, "frequency": "Daily", "id": "/lccn/sn83045433/1922-07-30/ed-1/seq-32/", "subject": ["Washington (D.C.)--fast--(OCoLC)fst01204505", "Washington (D.C.)--Newspapers."],**

If we were to pursue this research further, a next step might be cleaning the data to reduce the number of false positives, or find how many newspapers about fire appear on the front page over time.

**What users want from an API:**

1. **Documentation** is a user's starting point when working with a new API.
2. **Well designed URLs** make it serier for users to intuitively find resources/endpoints.

**Implementing our API**

1. Overview

* Build prototype API using **Python** and **Flask web framework.**
* Our example API will take the form of a **distant reading archive** - *a book catalog that goes beyond standard bibliographic information to include data of interest to those working on digital projects. Our API will serve the* ***title, date of publication, and first sentence of each book****.*

1. Create basic Flask app

* Why Flask?:

Python has a number of web frameworks to create web apps and APIs (Django). Flask applications tend to be written on a **blank canvas**, and are more suited to a contained application such as an API.

* Create a new folder on the computer that will be a project folder.
* Create a file called **api.py** in a **api folder**:

**import** flask

app **=** flask.Flask(\_\_name\_\_)

app.config["DEBUG"] **=** True

**@**app.route('/', methods**=**['GET'])

**def** **home**():

**return** "<h1>Distant Reading Archive</h1><p>This site is a prototype API for distant reading of science fiction novels.</p>"

app.run()

1. Running the app

* Navigate to the **api folder:**

cd ~/FinalYearProject/API\_research/API\_prototype/api

* Run the Flask application with command:

python api.py #activate the virtual env

* Expected output:

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

1. What Flask does

* Flask maps HTTP requests to Python functions.
  + In our case we mapped the **URL path “/”** to the function **“home”**.
* Process of mapping URLs to functions = **routing**.

@app.route('/', methods=['GET'])

* ‘GET’ = get data from application
* ‘POST’ = send data from user.
* **Component explanation - Run a Flask application:**
  + import flask
    - Import the Flask Library.
  + app = flask.Flask(\_\_name\_\_)
    - Create Flask app object, which contains data about the object and methods (object functions) that tell the app to do certain actions. (app.run() is one such method)
  + app.config["DEBUG"] = True
    - Start the debugger. If the ocde malformed, you'll see an error when visiting the app.
  + app.run()
    - Runs the application server.

1. Creating the API

* Add our **data as a list of Python dictionaries**.
* Key = type of information represented.
* Value = the actual data.

[{

'name': 'Alexander Graham Bell',

'number': '1-333-444-5555'

},

{

'name': 'Thomas A. Watson',

'number': '1-444-555-6666'

}

]

* Our example is a Phone Book with each dictionary a phone book entry consisting of ‘name’ and ‘number’.
* For our Flask API, we add data on three science fiction novels (id, title, author, first\_sentence, year\_published).
* **import** flask
* **from** flask **import** request, jsonify
* app **=** flask.Flask(\_\_name\_\_)
* app.config["DEBUG"] **=** True
* *# Create some test data for our catalog in the form of a list of dictionaries.*
* books **=** [
* {'id': 0,
* 'title': 'A Fire Upon the Deep',
* 'author': 'Vernor Vinge',
* 'first\_sentence': 'The coldsleep itself was dreamless.',
* 'year\_published': '1992'},
* {'id': 1,
* 'title': 'The Ones Who Walk Away From Omelas',
* 'author': 'Ursula K. Le Guin',
* 'first\_sentence': 'With a clamor of bells that set the swallows soaring, the Festival of Summer came to the city Omelas, bright-towered by the sea.',
* 'published': '1973'},
* {'id': 2,
* 'title': 'Dhalgren',
* 'author': 'Samuel R. Delany',
* 'first\_sentence': 'to wound the autumnal city.',
* 'published': '1975'}
* ]
* **@**app.route('/', methods**=**['GET'])
* **def** **home**():
* **return** '''<h1>Distant Reading Archive</h1>
* <p>A prototype API for distant reading of science fiction novels.</p>'''
* *# A route to return all of the available entries in our catalog.*
* **@**app.route('/api/v1/resources/books/all', methods**=**['GET'])
* **def** **api\_all**():
* **return** jsonify(books)
* app.run()

1. Finding Specific Resources

* Add functionality that allows users to **filter their returned data** using a more specific URL.
* Add following code:  
  **@**app.route('/api/v1/resources/books', methods**=**['GET'])
* **def** **api\_id**():
* *# Check if an ID was provided as part of the URL.*
* *# If ID is provided, assign it to a variable.*
* *# If no ID is provided, display an error in the browser.*
* **if** 'id' **in** request.args:
* id **=** int(request.args['id'])
* **else**:
* **return** "Error: No id field provided. Please specify an id."
* *# Create an empty list for our results*
* results **=** []
* *# Loop through the data and match results that fit the requested ID.*
* *# IDs are unique, but other fields might return many results*
* **for** book **in** books:
* **if** book['id'] **==** id:
* results.append(book)
* *# Use the jsonify function from Flask to convert our list of*
* *# Python dictionaries to the JSON format.*
* **return** jsonify(results)

1. Understanding our Updated API

* Created new function **api\_id()**
* **@app.route** used to map function to the path **/api/v1/resources/books**
* Examine the provided URL and check the **query parameters** (?id=0)
* Check the query parameter with **Python request.args**

**API Design Principles**

* Our next version will pull data from a database before providing it to a user.
* Also will take additional query parameters allowing the user to filter by other fields.

1. **Designing Requests**

* Modern APIs philosophy is **REST**:
  + Based on four methods of **HTTP protocol** -> actions performed on data in a **database**:

1. POST -> CREATE
2. GET -> READ
3. PUT -> UPDATE
4. DELETE -> DELETE

* How requests should be formatted:
  + Consider a poorly designed example of API endpoint:

http://api.example.com/getbook/10

* + First problem is **semantic**, in REST API our verbs are GET,POST,PUT,DELETE, nad are defined in the **Python request method** rather than the request URL.
  + The word ‘get’ *should not* appear in the request URL.
* Incorporating these principles:

<http://api.example.com/books/10>

* The above request uses part of the path, **/10**, to provide the ID. This is somewhat inflexible - generally only filtered by one field at a time.
* **Query Parameters (?id=10)** allow to filter multiple database fields.

http://api.example.com/books?author=Ursula+K.+Le Guin&published=1969&output=xml

* When designing how requests to your API should be structured, makes sense to **plan for future additions**. Even if the current API only serves one type of **resource**, you might add more resources in future, add **/resources/** to URL.

http://api.example.com/resources/books?id=10

* Another way to plan for the future is to add **version number** to the path **/v1/**.

https://api.example.com/v1/resources/books?id=10

1. **Documentation and Examples**

* Without documentation, even the best-designed API will be unusable.
* Your API should have:
  + *Describing the resources.*
    - *Section for each resource that describes with fields it accepts (id, title), in form of a sample HTTP request or block of code.*
  + *Describing the functionality available.*
  + *Concrete working examples of request URLs or code.*
* Common practice in documenting APIs is to provide **annotations**  in your code that are automatically collated into documentation using a tool such as **Doxygen** or **Sphinx**.
  + Tools create documentation from **docstrings** - comments you make on your function definitions.
  + Helpful but not the end of the documentation.
* Put yourself in the position of a potential user of your API and provide working examples.
* For inspiration on how to approach API documentation:

<http://api.repo.nypl.org/>

1. **Connecting API to Database**

* This example of our API, pulls data from a database, implements error handling, and can filter books by publication date.
* Database used is **SQLite**.

[download the example database](https://programminghistorian.org/assets/creating-apis-with-python-and-flask/books.db)

* Copy the file to the **api** folder.
* API code - **api\_final.py**:

**import** flask

**from** flask **import** request, jsonify

**import** sqlite3

app **=** flask.Flask(\_\_name\_\_)

app.config["DEBUG"] **=** True

**def** **dict\_factory**(cursor, row):

d **=** {}

**for** idx, col **in** enumerate(cursor.description):

d[col[0]] **=** row[idx]

**return** d

**@**app.route('/', methods**=**['GET'])

**def** **home**():

**return** '''<h1>Distant Reading Archive</h1>

<p>A prototype API for distant reading of science fiction novels.</p>'''

**@**app.route('/api/v1/resources/books/all', methods**=**['GET'])

**def** **api\_all**():

conn **=** sqlite3.connect('books.db')

conn.row\_factory **=** dict\_factory

cur **=** conn.cursor()

all\_books **=** cur.execute('SELECT \* FROM books;').fetchall()

**return** jsonify(all\_books)

**@**app.errorhandler(404)

**def** **page\_not\_found**(e):

**return** "<h1>404</h1><p>The resource could not be found.</p>", 404

**@**app.route('/api/v1/resources/books', methods**=**['GET'])

**def** **api\_filter**():

query\_parameters **=** request.args

id **=** query\_parameters.get('id')

published **=** query\_parameters.get('published')

author **=** query\_parameters.get('author')

query **=** "SELECT \* FROM books WHERE"

to\_filter **=** []

**if** id:

query **+=** ' id=? AND'

to\_filter.append(id)

**if** published:

query **+=** ' published=? AND'

to\_filter.append(published)

**if** author:

query **+=** ' author=? AND'

to\_filter.append(author)

**if** **not** (id **or** published **or** author):

**return** page\_not\_found(404)

query **=** query[:**-**4] **+** ';'

conn **=** sqlite3.connect('books.db')

conn.row\_factory **=** dict\_factory

cur **=** conn.cursor()

results **=** cur.execute(query, to\_filter).fetchall()

**return** jsonify(results)

app.run()

* Run with **python api\_final.py**.

1. **Understanding Database-powered API**

* Relational databases allow for storage and retrieval of data, stored in tables.
* Tables are similar to spreadsheets in that they have **columns and rows**.
  + **Columns = what the data represents** (title, date)
  + **Rows = individual entries** (books, users, any kind of entity)
* Our Database has five columns [id pulbished author title first\_sentence]
* Each row represents one book that won the Hugo award in the year under the [published] column.
* Our **api\_all** function pulls data from the Hugo database.

**def** **api\_all**():

conn **=** sqlite3.connect('books.db')

conn.row\_factory **=** dict\_factory

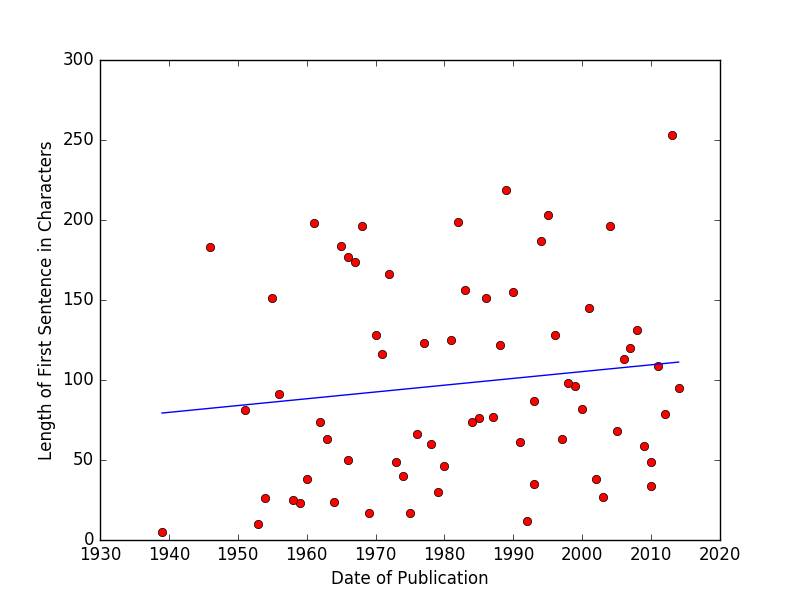
cur **=** conn.cursor()

all\_books **=** cur.execute('SELECT \* FROM books;').fetchall()

**return** jsonify(all\_books)

* Connect to DB using sqlite3 library.
* **conn** = object representing the connection to the database.
* **conn.row\_factory = dict\_factory** = lets connection object know to use the **dict\_factory** function we defined earlier. Which returns items from the DB as dictionaries rather than lists - these better for JSON output.
* **cur = conn.cursor()** = create cursor object, which is what moves through the database to pull our data.
* cur.execute = execute an SQL query, to pull out all available data **(\*)** from the **books table.**
* **jsonfiy(all\_books)** = data returned is JSON.

1. **API in practice**



Resources:

* API research <https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask>
* Endpoint def <https://stevenpcurtis.medium.com/endpoint-vs-api-ee96a91e88ca>