How To Use: API’s

Tools and resources used: Python Crash Course, Third Edition By Eric Matthes, Chat GPT 4.0

Before we do any of that we have to ensure we have the library requests installed onto our python directory.

API’s are a set of protocols with information from a website that can be used to monitor specific data. We use a URL to call the website's data, this link can filter out what kind of data we are looking for.

In the following code we call protocols and some of them are in Json format, others are in dictionary format, so according to what format our data is in we need to call the data either using Header or array allocation.

To format data for API we have to use dictionaries that we can loop information into. We can also use the plotly.express library to visualize the results of our data.

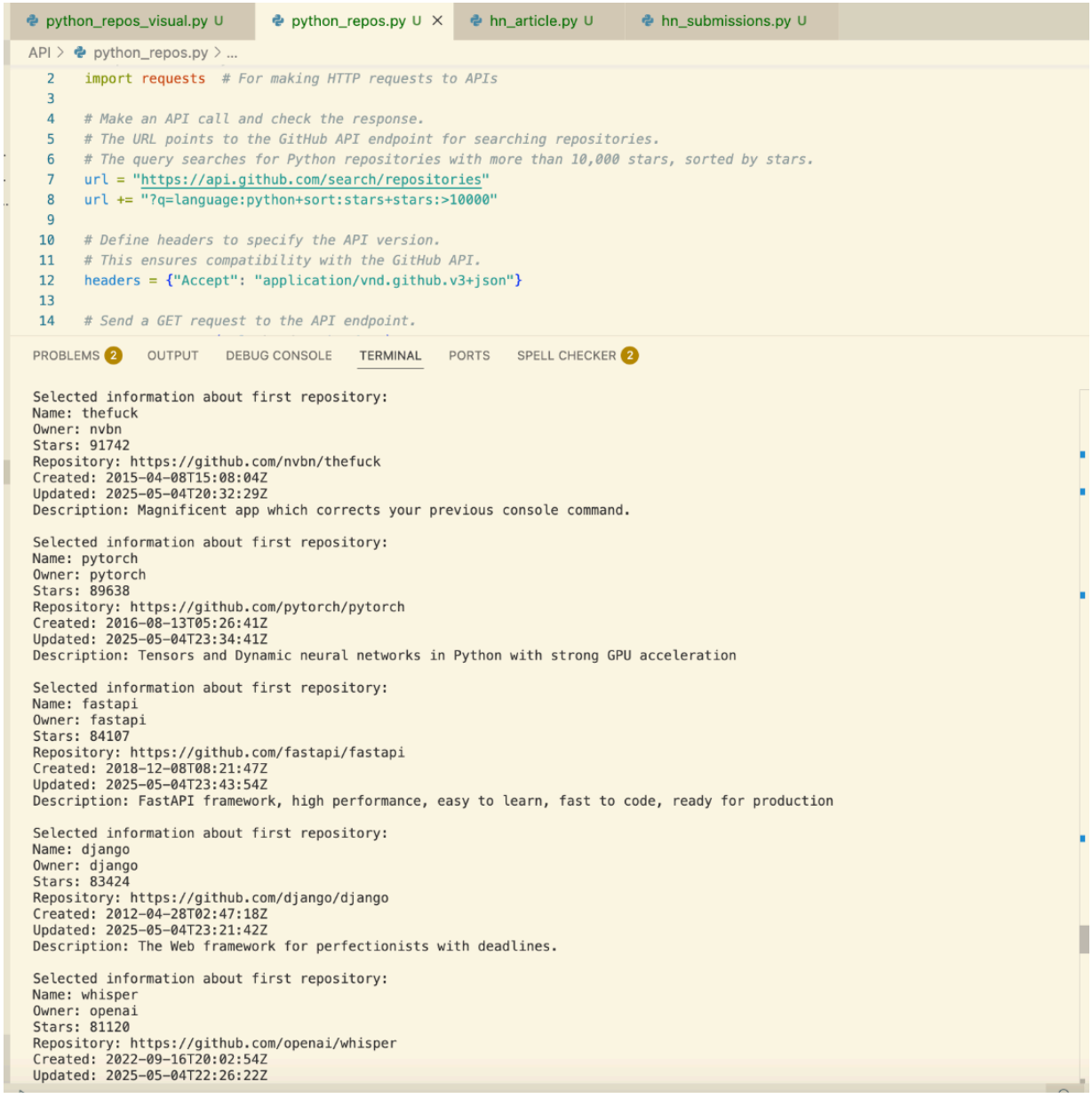
Here are a list of programs and their outputs:

Python_repos.py

```
# Import necessary libraries
import requests  # For making HTTP requests to APIs

# Make an API call and check the response.
# The URL points to the GitHub API endpoint for searching repositories.
# The query searches for Python repositories with more than 10,000 stars, sorted by
stars.
url = "https://api.github.com/search/repositories"
url += "?q=language:python+sort:stars+stars:>10000"

# Define headers to specify the API version.
# This ensures compatibility with the GitHub API.
headers = {"Accept": "application/vnd.github.v3+json"}

# Send a GET request to the API endpoint.
r = requests.get(url, headers=headers)

# Print the status code to ensure the request was successful (200 means OK).
print(f"Status code: {r.status_code}")

# Convert the response object to a dictionary.
```

```python
# The JSON response is converted into a Python dictionary for easier processing.
response_dict = r.json()

# Print the total number of repositories that match the query.
print(f"Total repositories: {response_dict['total_count']}")

# Check if the results are complete (not truncated).
# The 'incomplete_results' key indicates whether the results are truncated.
print(f"Complete results: {not response_dict['incomplete_results']}")

# Explore information about the repositories.
# Extract the list of repositories from the 'items' key in the response.
repo_dicts = response_dict['items']
print(f"Repositories returned: {len(repo_dicts)}")  # Print the number of repositories
returned.

# Loop through each repository and print selected information.
print("\nSelected information about each repository:")
for repo_dict in repo_dicts:
    # Print details about the repository.
    print("\nSelected information about first repository:")
    print(f"Name: {repo_dict['name']}")  # Repository name
    print(f"Owner: {repo_dict['owner']['login']}")  # Repository owner's username
    print(f"Stars: {repo_dict['stargazers_count']}")  # Number of stars
    print(f"Repository: {repo_dict['html_url']}")  # URL to the repository
    print(f"Created: {repo_dict['created_at']}")  # Creation date
    print(f"Updated: {repo_dict['updated_at']}")  # Last update date
    print(f"Description: {repo_dict['description']}")  # Repository description
```

Output:

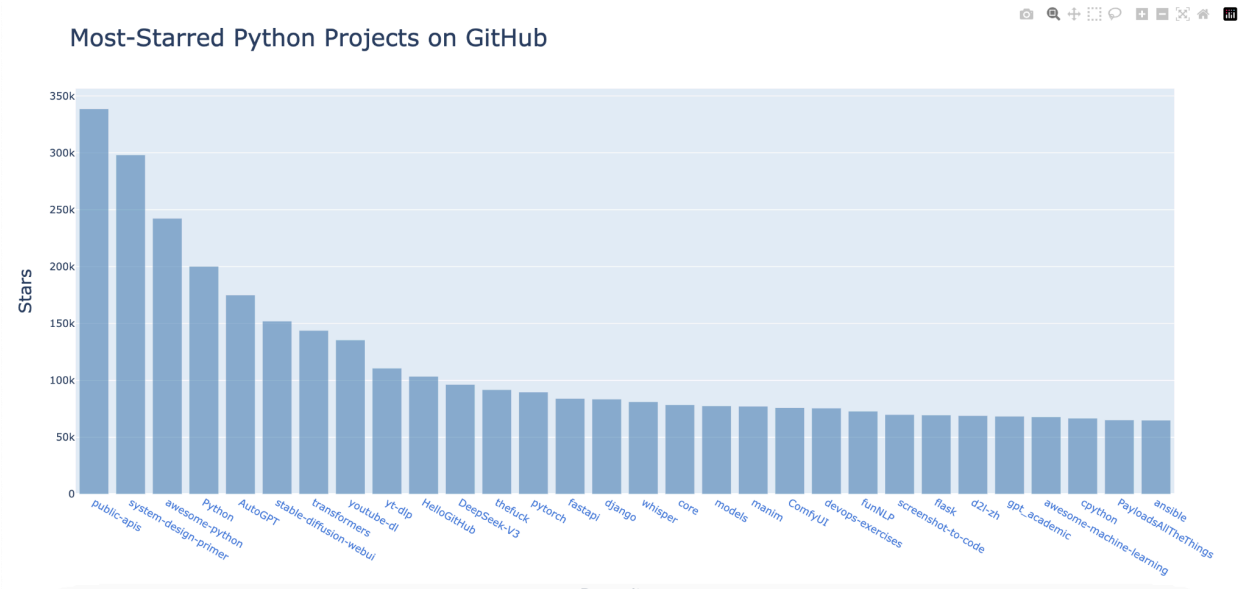

python_repos_visual.py U
python_repos.py U
hn_article.py U
hn_submissions.py U
API > python_repos.py > ...
2 import requests # For making HTTP requests to APIs
3
4 # Make an API call and check the response.
5 # The URL points to the GitHub API endpoint for searching repositories.
6 # The query searches for Python repositories with more than 10,000 stars, sorted by stars.
7 url = "https://api.github.com/search/repositories"
8 url += "?q=language:python+sort:stars+stars:>10000"
9
10 # Define headers to specify the API version.
11 # This ensures compatibility with the GitHub API.
12 headers = {"Accept": "application/vnd.github.v3+json"}
13
14 # Send a GET request to the API endpoint.
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SPELL CHECKER 2
Selected information about first repository:
Name: thefuck
Owner: nvbn
Stars: 91742
Repository: https://github.com/nvbn/thefuck
Created: 2015-04-08T15:08:04Z
Updated: 2025-05-04T20:32:29Z
Description: Magnificent app which corrects your previous console command.
Selected information about first repository:
Name: pytorch
Owner: pytorch
Stars: 89638
Repository: https://github.com/pytorch/pytorch
Created: 2016-08-13T05:26:41Z
Updated: 2025-05-04T23:34:41Z
Description: Tensors and Dynamic neural networks in Python with strong GPU acceleration
Selected information about first repository:
Name: fastapi
Owner: fastapi
Stars: 84107
Repository: https://github.com/fastapi/fastapi
Created: 2018-12-08T08:21:47Z
Updated: 2025-05-04T23:43:54Z
Description: FastAPI framework, high performance, easy to learn, fast to code, ready for production
Selected information about first repository:
Name: django
Owner: django
Stars: 83424
Repository: https://github.com/django/django
Created: 2012-04-28T02:47:18Z
Updated: 2025-05-04T23:21:42Z
Description: The Web framework for perfectionists with deadlines.
Selected information about first repository:
Name: whisper
Owner: openai
Stars: 81120
Repository: https://github.com/openai/whisper
Created: 2022-09-16T20:02:54Z
Updated: 2025-05-04T22:26:22Z

Python_repos_visual.py

```
# Import necessary libraries
import requests  # For making HTTP requests to APIs
import plotly.express as px  # For creating visualizations

# Make an API call and check the response.
# The URL points to the GitHub API endpoint for searching repositories.
# The query searches for Python repositories with more than 10,000 stars, sorted by
stars.
url = "https://api.github.com/search/repositories"
url += "?q=language:python+sort:stars+stars:>10000"

# Define headers to specify the API version.
# This ensures compatibility with the GitHub API.
headers = {"Accept": "application/vnd.github.v3+json"}

# Send a GET request to the API endpoint.
r = requests.get(url, headers=headers)

# Print the status code to ensure the request was successful (200 means OK).
print(f"Status code: {r.status_code}")

# Process the overall results.
# Convert the JSON response to a Python dictionary.
response_dict = r.json()

# Check if the results are complete (not truncated).
print(f"Complete results: {not response_dict['incomplete_results']}")

# Process repository information.
# Extract the list of repositories from the 'items' key in the response.
repo_dicts = response_dict['items']

# Initialize lists to store data for the visualization.
repo_links, stars, hover_texts = [], [], []

# Loop through each repository in the list.
for repo_dict in repo_dicts:
    # Turn repository names into active links.
    # This creates a clickable link for each repository.
    repo_name = repo_dict['name']
```

```
    repo_url = repo_dict['html_url']
    repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
    repo_links.append(repo_link)

    # Append the number of stars for each repository.
    stars.append(repo_dict['stargazers_count'])

    # Build hover texts for each repository.
    # Include the owner's username and the repository description.
    owner = repo_dict['owner']['login']
    description = repo_dict['description']
    hover_text = f"{owner}<br />{description}"
    hover_texts.append(hover_text)

# Make the visualization.
# Create a bar chart using Plotly Express.
title = "Most-Starred Python Projects on GitHub"  # Set the chart title.
labels = {'x': 'Repository', 'y': 'Stars'}  # Define axis labels.
fig = px.bar(
    x=repo_links,  # Use repository links for the x-axis.
    y=stars,  # Use the number of stars for the y-axis.
    title=title,  # Set the chart title.
    labels=labels,  # Set axis labels.
    hover_name=hover_texts  # Display hover text for each bar.
)

# Customize the layout of the chart.
fig.update_layout(
    title_font_size=28,  # Set the font size for the title.
    xaxis_title_font_size=20,  # Set the font size for the x-axis label.
    yaxis_title_font_size=20  # Set the font size for the y-axis label.
)

# Customize the appearance of the bars.
fig.update_traces(
    marker_color='SteelBlue',  # Set the bar color to SteelBlue.
    marker_opacity=0.6  # Set the bar opacity to 60%.
)

# Display the chart.
fig.show()
```

Output:



Hn_article.py

```python
# Import necessary libraries
import requests  # For making HTTP requests to APIs
import json  # For working with JSON data

# Make an API call, and store the response.
# The URL points to a specific Hacker News article in JSON format.
url = "https://hacker-news.firebaseio.com/v0/item/31353677.json"
r = requests.get(url)  # Send a GET request to the API endpoint
print(f"Status code: {r.status_code}")  # Print the status code to ensure the request
was successful

# Explore the structure of the data.
# Convert the JSON response to a Python dictionary.
response_dict = r.json()

# Format the dictionary as a JSON string with indentation for readability.
response_string = json.dumps(response_dict, indent=4)

# Print the formatted JSON string to explore the structure of the data.
print(response_string)
```

Output:

```
Complete results: True
joseochoa@Mac Projects %  cd /Users/joseochoa/Desktop/Python\ Coding\ Class/Projects ; /usr/bin/env /usr/local/bin/python3 /Users
/joseochoa/.vscode/extensions/ms-python.debugpy-2025.6.0-darwin-arm64/bundled/libs/debugpy/adapter/../../debugpy/launcher 57725 -
- /Users/joseochoa/Desktop/Python\ Coding\ Class/Projects/API/hn_article.py
Status code: 200
{
    "by": "sohkamyung",
    "descendants": 307,
    "id": 31353677,
    "kids": [
        31354987,
        31354235,
        31354040,
        31358602,
        31354201,
        31354991,
        31354315,
        31353775,
        31353925,
        31354169,
        31354273,
        31354437,
        31356902,
        31358694,
        31363418,
        31353862,
        31357186,
        31356379,
        31356826,
        31355085,
        31369435,
        31357936,
        31354142,
        31354213,
        31356311,
        31357865,
        31353929,
        31364954,
```

Hn_submissions.py

```python
from operator import itemgetter  # For sorting dictionaries by a specific key
import requests  # For making HTTP requests to APIs

# Make an API call to get the top stories on Hacker News.
# The URL points to the top stories endpoint, which returns a list of submission IDs.
url = 'https://hacker-news.firebaseio.com/v0/topstories.json'
r = requests.get(url)  # Send a GET request to the API endpoint
print(f"Status code: {r.status_code}")  # Print the status code to ensure the request
was successful

# Process the list of submission IDs returned by the API.
# The response is a JSON array of IDs, which we convert to a Python list.
submission_ids = r.json()

# Initialize a list to store information about each submission.
submission_dicts = []

# Loop through the first 30 submission IDs.
# For each ID, make a new API call to get details about the submission.
for submission_id in submission_ids[:30]:
```

```
    # Construct the URL for the specific submission.
    url = f"https://hacker-news.firebaseio.com/v0/item/{submission_id}.json"
    r = requests.get(url)  # Send a GET request to the API endpoint
    print(f"id: {submission_id}\tstatus: {r.status_code}")  # Print the ID and status
code for debugging
    response_dict = r.json()  # Convert the JSON response to a Python dictionary

    # Build a dictionary for each article.
    # Use the `.get()` method to safely access keys and provide default values if keys
are missing.
    submission_dict = {
        'title': response_dict.get('title', 'No title'),  # Get the title or use 'No
title' if missing
        'hn_link': f"https://news.ycombinator.com/item?id={submission_id}",  #
Construct the discussion link
        'comments': response_dict.get('descendants', 0),  # Get the number of comments
or default to 0
    }
    submission_dicts.append(submission_dict)  # Add the dictionary to the list

# Sort the list of submissions by the number of comments in descending order.
# Use `itemgetter` to specify the 'comments' key for sorting.
submission_dicts = sorted(submission_dicts, key=itemgetter('comments'), reverse=True)

# Print the top submissions.
# Loop through the sorted list and print the title, discussion link, and number of
comments for each submission.
for submission_dict in submission_dicts:
    print(f"\nTitle: {submission_dict['title']}")  # Print the title of the submission
    print(f"Discussion link: {submission_dict['hn_link']}")  # Print the discussion
link
    print(f"Comments: {submission_dict['comments']}")  # Print the number of comments
```

Output:


PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SPELL CHECKER 2
Comments: 19
Title: Typed Lisp, a Primer
Discussion link: https://news.ycombinator.com/item?id=43887998
Comments: 15
Title: Minimal Linux Bootloader (2018)
Discussion link: https://news.ycombinator.com/item?id=43887708
Comments: 15
Title: The Speed of VITs and CNNs
Discussion link: https://news.ycombinator.com/item?id=43866329
Comments: 13
Title: Tech Companies Apparently Do Not Understand Why We Dislike AI
Discussion link: https://news.ycombinator.com/item?id=43890502
Comments: 8
Title: Graceful Shutdown in Go: Practical Patterns
Discussion link: https://news.ycombinator.com/item?id=43889610
Comments: 6
Title: Show HN: Driverless print server for legacy printers, profit goes to open-source
Discussion link: https://news.ycombinator.com/item?id=43888157
Comments: 6
Title: Cyborg cicadas play Pachelbel's Canon
Discussion link: https://news.ycombinator.com/item?id=43882287
Comments: 5
Title: Orders of Infinity
Discussion link: https://news.ycombinator.com/item?id=43888239
Comments: 5
Title: Load-Store Conflicts
Discussion link: https://news.ycombinator.com/item?id=43888005
Comments: 5
Title: Show HN: EZ-TRAK Satellite Hand Tracking Suite
Discussion link: https://news.ycombinator.com/item?id=43887546
Comments: 5
Title: Thunderscope update: My take: Why open source is better