
🧠 Assignment: Meeting Video Scraper

Welcome! This task involves developing a robust scraping system for collecting and downloading public meeting videos. There are **two main problems** and **one bonus task**.

📌 Problem 1: Scraping Meeting Metadata

🧩 Objective

Scrape meeting metadata (video title, date, link, etc.) from a list of public video hosting websites over a specified date range.

🗒️ Input

A JSON object containing:

- `start_date`: The start date of the range (inclusive)
- `end_date`: The end date of the range (inclusive)
- `base_urls`: A list of public video directory base URLs

✅ Expected Output

For each `base_url`, return all matching meeting metadata:

```
[
  {
    "base_url": "http://detroit-vod.cablecast.tv/CablecastPublicSite",
    "medias": [
      {
        "url": "http://detroit-vod.cablecast.tv/CablecastPublicSite/show/14093?site=1",
        "title": "Detroit City Council Formal Session pt2 11-26-2024",
        "date": "2024-11-26",
        "source_type": "video"
      },
      ...
    ]
  }
]
```

⚠️ Metadata must be filtered based on the `start_date` and `end_date`.

📂 Sample Input

```
{
  "start_date": "2024-11-20",
  "end_date": "2024-11-26",
  "base_urls": ["http://detroit-vod.cablecast.tv/CablecastPublicSite"]
}
```

📁 Testing Data

Base_urls:

https://www.lansdale.org/CivicMedia?CID=2024-Council-Meetings-26
https://www.facebook.com/DauphinCountyPA/videos
https://charlestonwv.portal.civicclerk.com/
https://www.youtube.com/@SLCLiveMeetings/streams

https://www.regionalwebtv.com/fredcc

https://winchesterva.civicweb.net/portal/

🎯 Problem 2: Video Download URL Resolution

🧩 Objective

Extract direct video URLs from scraped meeting pages to be used by `yt-dlp` for downloading.

Some links will be webpage URLs containing embedded players. You need to:

- Visit the webpage
- Locate and extract the downloadable video source URL
- Verify that it works with `yt-dlp`

🗒️ Input

A list of meeting URLs (from Problem 1).

✅ Expected Output

Return only the video URLs that can be downloaded using `yt-dlp`.

```
[
  "https://dallastx.new.swagit.com/videos/320946/download"
]
```

> 💡 Use `yt-dlp --simulate` to check if a URL is valid without downloading.

📂 Sample Input

```
[
  "https://dallastx.new.swagit.com/videos/320946"
]
```

📁 Testing Data

URLs:

https://www.zoomgov.com/rec/share/vCZnQM5bgMzY7_n4lbQXYnVqsvPj49Ce-R0hMjMFPyG4FUC1HbSyQAZ9uPpRKDvV._6ZMrf7BXZzx6_RK?startTime=1709655569000

https://snohomish.legistar.com/Video.aspx?Mode=Granicus&ID1=9188&Mode2=Video

https://multnomah.granicus.com/MediaPlayer.php?view_id=3&clip_id=3097
https://traviscotx.portal.civicclerk.com/event/2583/media
https://legistar.council.nyc.gov/Video.aspx?Mode=Auto&URL=aHR0cHM6Ly9jb3VuY2lsbnljLnZpZWJpdC5jb20vdm9kLz9zPXRydWUmdj1OWUNDLVBWLTI1MC0xNF8yNDA2MDQtMTAxMzU0Lm1wNA%3d%3d&Mode2=Video

https://video.ibm.com/recorded/134312408

https://cityofalhambraorg-my.sharepoint.com/:v:/g/personal/lmyles_alhambraca_gov/ETs6K1euPsBClaWtczJXl-gB47R9yoz9o9FJYZuEY0KOjA?e=7B41Fy

https://audiomack.com/pemberton-twp-planningzoning-board-meetings/song/678668c3069f2

Special URLs (Bonus Points if you can solve those):
https://video.ibm.com/recorded/134312408

https://monticello.viebit.com/watch?hash=HCZTN4vuyJ91LlrS

https://play.champds.com/guilderlandny/event/431

💡 Bonus Task: Faster Downloads with `aria2c`

🧩 Objective

Improve the download speed of videos using external downloaders like `aria2c`.

📚 Background

`yt-dlp` can integrate with `aria2c` to enable multi-threaded and faster downloads. This is done by passing the `--external-downloader aria2c` flag.

✅ Task Requirements

- Convert your download process to use `yt-dlp` **programmatically via Python**
- Configure `aria2c` with options like multiple connections, split parts, and optimized retry policies

---

🏅 Evaluation Criteria:

- Correctness (40%): Does the scraper accurately extract and filter metadata and video URLs?
- Robustness (20%): Can the scraper handle edge cases, broken links, and varied website structures?
- Efficiency (20%): Is the scraping and downloading process optimized for speed?
- Code Quality (10%): Is the code well-structured, modular, and reusable?
- Documentation (10%): Are functions and usage clearly documented?

🚀 Submission Instructions:

- Create a GitHub repository to host your solution.
- Provide clear documentation (`README.md`) explaining how to run your scraper.
- Include examples and outputs.
- Clearly highlight the bonus task if completed.
- Make a 5-10 minutes demo video or loom, walk through your codebase and your findings and add it to readme.
- Feel free to use AI tools to support your coding process—just ensure clarity and correctness.

🧪 Final Notes

- You are encouraged to build reusable functions and modularize the code
- Assume this scraper will be used at scale later
- Make sure the output is clean, structured JSON ready for ingestion into a database
- For the bonus task it'll be awesome to know how much faster you can make it

Happy coding!