Daily Pennsylvanian Sports Headline Scraper

claude.ai/chat/12f2ebff-2e0f-4aec-b31f-e233ab0225c3

AG

""" Scrapes a headline from The Daily Pennsylvanian website and saves it to a JSON file that tracks headlines over time. """ import os import sys import daily_event_monitor import bs4 import requests import loguru def scrape_data_point(): """ Scrapes the main headline from The Daily Pennsylvanian home page. Returns: str: The headline text if found, otherwise an empty string. """ headers = { "User-Agent": "cis3500-scraper" } req = requests.get("https://www.thedp.com", headers=headers) loguru.logger.info(f"Request URL: {req.url}") loguru.logger.info(f"Request status code: {req.status_code}") if req.ok: soup = bs4.BeautifulSoup(req.text, "html.parser") target*element = soup.find("a", class*="frontpage-link") data_point = "" if target_element is None else target_element.text loguru.logger.info(f"Data point: {data_point}") return data_point if **name** == "main": # Setup logger to track runtime loguru.logger.add("scrape.log", rotation="1 day") # Create data dir if needed loguru.logger.info("Creating data directory if it does not exist") try: os.makedirs("data", exist_ok=True) except Exception as e: loguru.logger.error(f"Failed to create data directory: {e}") sys.exit(1) # Load daily event monitor loguru.logger.info("Loading daily event monitor") dem = daily_event_monitor.DailyEventMonitor("data/daily_pennsylvanian_headlines.json") # Run scrape loguru.logger.info("Starting scrape") try: data_point = scrape_data_point() except Exception as e: loguru.logger.error(f"Failed to scrape data point: {e}") data_point = None # Save data if data_point is not None: dem.add_today(data_point) dem.save() loguru.logger.info("Saved daily event monitor") def print_tree(directory, ignore_dirs=[".git", "pycache"]): loguru.logger.info(f"Printing tree of files/dirs at {directory}") for root, dirs, files in os.walk(directory): dirs[:] = [d for d in dirs if d not in ignore_dirs] level = root.replace(directory, "").count(os.sep) indent = " " * 4 * (level) loguru.logger.info(f"{indent}+--{os.path.basename(root)}/") sub_indent = " " * 4 * (level + 1) for file in files: loguru.logger.info(f"{sub_indent}+--{file}") print_tree(os.getcwd()) loguru.logger.info("Printing contents of data file {}".format(dem.file_path)) with open(dem.file_path, "r") as f: loguru.logger.info(f.read()) # Finish loguru.logger.info("Scrape complete") loguru.logger.info("Exiting")

So this file is called script.py and its part of a repsitory called daily Pennsylvania scraper. Can you explain me the code function by function

I'll break down each function and section of the code for you:

1. First, let's look at `scrape_data_point()`:

python

```python
def scrape_data_point():
    headers = {
        "User-Agent": "cis3500-scraper"
    }
    req = requests.get("https://www.thedp.com", headers=headers)
    loguru.logger.info(f"Request URL: {req.url}")
    loguru.logger.info(f"Request status code: {req.status_code}")
```

This function:

- Makes a web request to The Daily Pennsylvanian website
- Uses a custom User-Agent header to identify the scraper
- Logs the URL and response status code
- Then parses the response to find the main headline:

```
python
```

```
  if req.ok:
      soup = bs4.BeautifulSoup(req.text, "html.parser")
      target_element = soup.find("a", class_="frontpage-link")
      data_point = "" if target_element is None else target_element.text
      loguru.logger.info(f"Data point: {data_point}")
      return data_point
```

2. The `print_tree()` function:

```
python
```

```
def print_tree(directory, ignore_dirs=[".git", "__pycache__"]):
```

This is a utility function that:

- Walks through a directory structure
- Prints a visual tree representation of files and folders
- Ignores specified directories like .git and **pycache**
- Uses indentation to show hierarchy
- Logs the structure using loguru

3. The main execution block:

```
python
```

```
if __name__ == "__main__":
```

This section runs when the script is executed directly and:

a. Sets up logging:

```python
loguru.logger.add("scrape.log", rotation="1 day")
```

Creates a log file that rotates daily

b. Creates data directory:

```python
try:
    os.makedirs("data", exist_ok=True)
```

- Makes a 'data' directory if it doesn't exist
- Exits if directory creation fails

c. Initializes the daily event monitor:

```python
dem = daily_event_monitor.DailyEventMonitor(
    "data/daily_pennsylvanian_headlines.json"
)
```

Creates/loads a JSON file to store headlines

d. Runs the scraping operation:

python

```
try:
    data_point = scrape_data_point()
except Exception as e:
    loguru.logger.error(f"Failed to scrape data point: {e}")
    data_point = None
```

- Attempts to scrape the headline
- Handles any errors that occur

e. Saves the data:

python

```
if data_point is not None:
    dem.add_today(data_point)
    dem.save()
```

If scraping was successful, saves the headline

f. Finally, it:

- Prints the directory structure
- Shows the contents of the data file
- Logs completion

The script uses several external dependencies:

- `requests` for making HTTP requests
- `bs4` (BeautifulSoup4) for parsing HTML
- `loguru` for logging
- `daily_event_monitor` (appears to be a custom module) for managing the headline data

Would you like me to explain any particular part in more detail?

Claude does not have the ability to run the code it generates yet.

AG

soup = bs4.BeautifulSoup(req.text, "html.parser") target*element = soup.find("a", class*="frontpage-link")

this is finding the html elements to scrap I assume. If I wanna scrap a different one how should I modify my code

Yes, exactly! That code is using BeautifulSoup to parse the HTML and find specific elements. To scrape different elements, you'll need to:

1. First inspect the webpage's HTML to find the correct elements/selectors you want to target
2. Then modify the BeautifulSoup `find()` or use other BeautifulSoup methods

Here are some common ways to find elements:

python

```python
# Find by HTML tag and class
element = soup.find("div", class_="some-class")

# Find by ID
element = soup.find(id="some-id")

# Find by CSS selector
element = soup.select_one("#some-id .some-class")

# Find multiple elements
elements = soup.find_all("div", class_="article-preview")

# Find by multiple attributes
element = soup.find("div", attrs={"class": "title", "data-type": "news"})

# Find nested elements
element = soup.find("div", class_="container").find("span", class_="author")
```

Some other useful BeautifulSoup methods:

python

```
# Get text content
text = element.text  # Gets all text
text = element.get_text(strip=True)  # Gets text without extra whitespace

# Get specific attributes
href = element["href"]  # Get href attribute
href = element.get("href")  # Safer way to get attribute

# Get parent/siblings
parent = element.parent
next_sibling = element.next_sibling
```

For The Daily Pennsylvanian specifically, if you wanted to scrape different elements, you might modify the code like:

python

```
def scrape_data_point():
    headers = {"User-Agent": "cis3500-scraper"}
    req = requests.get("https://www.thedp.com", headers=headers)

    if req.ok:
        soup = bs4.BeautifulSoup(req.text, "html.parser")

        # Example: Get all article headlines
        headlines = soup.find_all("h2", class_="article-title")

        # Example: Get article text and author
        article = {
            "title": soup.find("h1", class_="article-title").text.strip(),
            "author": soup.find("span", class_="author-name").text.strip(),
            "content": soup.find("div", class_="article-content").text.strip()
        }

        # Example: Get links from the navigation menu
        nav_links = soup.select("nav.main-navigation a")

        return [h.text.strip() for h in headlines]  # or return article,
nav_links, etc.
```

Pro tip: To find the right selectors:

1. Open the webpage in Chrome/Firefox
2. Right-click the element you want to scrape
3. Click "Inspect" to see the HTML
4. Look for unique identifiers (ids, classes, or HTML structure)

Would you like me to help you target specific elements from The Daily Pennsylvanian website?

ⓘ
Claude does not have the ability to run the code it generates yet.

AG

if data_point is not None: dem.add_today(data_point) dem.save()

does this part of the code save the data to a json file?

This code doesn't directly save to JSON - it uses the `DailyEventMonitor` class (imported as `dem`) which handles the actual JSON file operations. Based on the code, the `DailyEventMonitor` class appears to be from a custom module called `daily_event_monitor` that manages saving headlines with their dates.

The process works like this:

1. `dem.add_today(data_point)` likely adds the headline (`data_point`) with today's date to some internal data structure
2. `dem.save()` then probably writes that data structure to the JSON file specified when creating the monitor: `"data/daily_pennsylvanian_headlines.json"`

If you want to see how the JSON file operations actually work, you would need to check the code in the `daily_event_monitor.py` file. But the result would likely look something like this:

```
json
```

```
{
  "2025-02-22": "Today's headline text",
  "2025-02-21": "Yesterday's headline text",
  "2025-02-20": "Headline from two days ago"
}
```

Would you like me to show you how to directly save data to a JSON file without using the DailyEventMonitor class? That might be useful if you want to modify how the data is stored.

ⓘ
Claude does not have the ability to run the code it generates yet.

AG

Yes that would be great

Here's how you could modify the code to directly save the scraped headlines to a JSON file:

```python
import json
from datetime import datetime

def save_to_json(data_point, filepath="data/headlines.json"):
    """
    Save a headline with today's date to a JSON file.
    If the file exists, it will update it; if not, it will create it.
    """
    # Get today's date as string
    today = datetime.now().strftime("%Y-%m-%d")

    try:
        # Try to read existing data
        with open(filepath, 'r') as f:
            data = json.load(f)
    except FileNotFoundError:
        # If file doesn't exist, start with empty dict
        data = {}

    # Add new data point with today's date
    data[today] = data_point

    # Save updated data back to file
    with open(filepath, 'w') as f:
        json.dump(data, f, indent=4)
```

Then you could replace the DailyEventMonitor part of your script with:

python

```python
# Run scrape
loguru.logger.info("Starting scrape")
try:
    data_point = scrape_data_point()
except Exception as e:
    loguru.logger.error(f"Failed to scrape data point: {e}")
    data_point = None

# Save data
if data_point is not None:
    try:
        save_to_json(data_point, "data/headlines.json")
        loguru.logger.info("Saved headline to JSON file")
    except Exception as e:
        loguru.logger.error(f"Failed to save to JSON: {e}")
```

This will create a JSON file that looks like:

json

```json
{
    "2025-02-22": "Today's headline",
    "2025-02-21": "Yesterday's headline",
    "2025-02-20": "Previous headline"
}
```

If you want to store more information with each headline, you could modify it like this:

python

```
def save_to_json(data_point, filepath="data/headlines.json"):
    today = datetime.now().strftime("%Y-%m-%d")
    timestamp = datetime.now().strftime("%H:%M:%S")

    try:
        with open(filepath, 'r') as f:
            data = json.load(f)
    except FileNotFoundError:
        data = {}

    # Store headline with additional metadata
    data[today] = {
        "headline": data_point,
        "time_scraped": timestamp,
        "source_url": "https://www.thedp.com"
    }

    with open(filepath, 'w') as f:
        json.dump(data, f, indent=4)
```

This would create a more detailed JSON structure:

json

```
{
    "2025-02-22": {
        "headline": "Today's headline text",
        "time_scraped": "14:30:25",
        "source_url": "https://www.thedp.com"
    },
    "2025-02-21": {
        "headline": "Yesterday's headline text",
        "time_scraped": "14:30:25",
        "source_url": "https://www.thedp.com"
    }
}
```

Would you like me to show you how to read the data back from the JSON file as well?

ⓘ
Claude does not have the ability to run the code it generates yet.

AG

""" Scrapes a headline from The Daily Pennsylvanian website and saves it to a JSON file that tracks headlines over time. """ import os import sys import daily_event_monitor import bs4 import requests import loguru def scrape_data_point(): """ Scrapes the main headline from The Daily Pennsylvanian home page. Returns: str: The headline text if found, otherwise an empty string. """ headers = { "User-Agent": "cis3500-scraper" } req = requests.get("https://www.thedp.com/section/sports", headers=headers) loguru.logger.info(f"Request URL: {req.url}") loguru.logger.info(f"Request status code: {req.status_code}") if req.ok: soup = bs4.BeautifulSoup(req.text, "html.parser") targete*lement = soup.find("h3", class*="standard_link") data_point = "" if target_element is None else target_element.text loguru.logger.info(f"Data point: {data_point}") return data_point if **name** == "main": # Setup logger to track runtime loguru.logger.add("scrape.log", rotation="1 day") # Create data dir if needed loguru.logger.info("Creating data directory if it does not exist") try: os.makedirs("data", exist_ok=True) except Exception as e: loguru.logger.error(f"Failed to create data directory: {e}") sys.exit(1) # Load daily event monitor loguru.logger.info("Loading daily event monitor") dem = daily_event_monitor.DailyEventMonitor("data/daily_pennsylvanian_headlines.json") # Run scrape loguru.logger.info("Starting scrape") try: data_point = scrape_data_point() except Exception as e: loguru.logger.error(f"Failed to scrape data point: {e}") data_point = None # Save data if data_point is not None: dem.add_today(data_point) dem.save() loguru.logger.info("Saved daily event monitor") def print_tree(directory, ignore_dirs=[".git", "pycache"]): loguru.logger.info(f"Printing tree of files/dirs at {directory}") for root, dirs, files in os.walk(directory): dirs[:] = [d for d in dirs if d not in ignore_dirs] level = root.replace(directory, "").count(os.sep) indent = " " * 4 * (level) loguru.logger.info(f"{indent}+--{os.path.basename(root)}/") sub_indent = " " * 4 * (level + 1) for file in files: loguru.logger.info(f"{sub_indent}+--{file}") print_tree(os.getcwd()) loguru.logger.info("Printing contents of data file {}".format(dem.file_path)) with open(dem.file_path, "r") as f: loguru.logger.info(f.read()) # Finish loguru.logger.info("Scrape complete") loguru.logger.info("Exiting")

i changed script.py to scrap sport headlines. Does this look good to you?

I see you made two key changes to scrape sports headlines:

1. Changed the URL from "https://www.thedp.com" to "https://www.thedp.com/section/sports"

2. Modified the target element from `find("a", class_="frontpage-link")` to `find("h3", class_="standard_link")`

The code changes look generally correct, but I'd suggest a couple of improvements:

1. You might want to handle multiple sports headlines instead of just one. Here's how you could modify the scrape_data_point function:

```
python
```

```
def scrape_data_point():
    headers = {
        "User-Agent": "cis3500-scraper"
    }
    req = requests.get("https://www.thedp.com/section/sports", headers=headers)
    loguru.logger.info(f"Request URL: {req.url}")
    loguru.logger.info(f"Request status code: {req.status_code}")
    if req.ok:
        soup = bs4.BeautifulSoup(req.text, "html.parser")
        # Find all sports headlines
        target_elements = soup.find_all("h3", class_="standard_link")
        # Get the first headline or empty string if none found
        data_point = "" if not target_elements else
target_elements[0].text.strip()
        loguru.logger.info(f"Data point: {data_point}")
        return data_point
```

2. It would be good to verify that you're actually getting sports headlines by testing the code first. You could add some debug logging:

python

```
if req.ok:
    soup = bs4.BeautifulSoup(req.text, "html.parser")
    target_element = soup.find("h3", class_="standard_link")
    if target_element is None:
        loguru.logger.warning("No sports headline found with class
'standard_link'")
    data_point = "" if target_element is None else target_element.text.strip()
    loguru.logger.info(f"Data point: {data_point}")
    return data_point
```

Have you tested the code to confirm it's successfully scraping sports headlines? Also, would you like me to show you how to scrape multiple sports headlines at once?

Claude does not have the ability to run the code it generates yet. Claude does not have internet access. Links provided may not be accurate or up to date.

AG

Yes do I need to use a loop because there is multiple deadline. How do I fetch the top one vs all of them

Yes, I'll show you how to fetch either the top headline or all headlines. Here are two versions of the scrape_data_point function:

python

```
# Version 1: Get just the top sports headline
def scrape_data_point():
    headers = {
        "User-Agent": "cis3500-scraper"
    }
    req = requests.get("https://www.thedp.com/section/sports", headers=headers)
    loguru.logger.info(f"Request URL: {req.url}")
    loguru.logger.info(f"Request status code: {req.status_code}")

    if req.ok:
        soup = bs4.BeautifulSoup(req.text, "html.parser")
        # Find first sports headline
        target_element = soup.find("h3", class_="standard_link")
        data_point = "" if target_element is None else
target_element.text.strip()
        loguru.logger.info(f"Top sports headline: {data_point}")
        return data_point

# Version 2: Get all sports headlines
def scrape_all_headlines():
    headers = {
        "User-Agent": "cis3500-scraper"
    }
    req = requests.get("https://www.thedp.com/section/sports", headers=headers)
    loguru.logger.info(f"Request URL: {req.url}")
    loguru.logger.info(f"Request status code: {req.status_code}")

    headlines = []
    if req.ok:
        soup = bs4.BeautifulSoup(req.text, "html.parser")
        # Find all sports headlines
        headline_elements = soup.find_all("h3", class_="standard_link")

        for element in headline_elements:
            headlines.append(element.text.strip())

        loguru.logger.info(f"Found {len(headlines)} sports headlines")
        loguru.logger.debug(f"Headlines: {headlines}")

    return headlines
```

To use these in your main script, you could do:

python

```
# For just top headline (current approach):
try:
    data_point = scrape_data_point()
except Exception as e:
    loguru.logger.error(f"Failed to scrape data point: {e}")
    data_point = None

# OR for all headlines:
try:
    headlines = scrape_all_headlines()
    if headlines:
        # Save first headline
        data_point = headlines[0]
        # Or save all headlines
        for headline in headlines:
            dem.add_today(headline)
    else:
        data_point = None
except Exception as e:
    loguru.logger.error(f"Failed to scrape headlines: {e}")
    data_point = None
```

The main differences are:

- `find()` gets just the first matching element
- `find_all()` gets all matching elements and returns a list
- With `find_all()` you need to loop through the results to process each headline

Would you like me to show you how to modify the JSON storage to handle multiple headlines per day as well?

ⓘ
Claude does not have the ability to run the code it generates yet.

AG

# Modified Basic Git Scraper For this git scraper, I used the daily pennsylvania site to scrap the top sports deadline everyday. In order to implement this firstly I had to import the original git scraper which can be found here https://github.com/jlumbroso/basic-git-scraper-template This scraper was fetching the main headline everyday. I modified the script.py file to go to sport section by modifying the URL from "https://www.thedp.com" to "https://www.thedp.com/section/sports". Then I also had to change the html element I was looking for from find("a", class=*"frontpage-link") to find("h3", class*="standard_link"). This was I was able to fetch the top sports deadline everyday.

can you imporve this read.me file

Here's an improved version of your README.md with better structure, more details, and clearer explanations:

A Python-based web scraper that automatically collects the top sports headline daily from The Daily Pennsylvanian's sports section.

Overview

This project modifies the basic-git-scraper-template to specifically track sports coverage from Penn's student newspaper. While the original scraper collected main headlines, this version focuses on sports content, providing a way to monitor Penn's athletic coverage over time.

Modifications Made

1. **URL Modification**:
    - Original: `https://www.thedp.com`
    - Modified: `https://www.thedp.com/section/sports`
2. **HTML Element Targeting**:
    - Original: `find("a", class_="frontpage-link")`
    - Modified: `find("h3", class_="standard_link")`

Key Features

- Daily automated scraping of top sports headlines
- JSON-based storage for historical tracking
- Logging system for monitoring scraper performance
- Error handling for reliable operation

Technical Implementation

The scraper uses:

- BeautifulSoup4 for HTML parsing

- Requests library for web scraping
- Loguru for comprehensive logging
- JSON for data storage

Getting Started

1. Clone this repository
2. Install requirements:

   ```
   bash
   ```

   ```
   pip install -r requirements.txt
   ```

3. Run the scraper:

   ```
   bash
   ```

   ```
   python script.py
   ```

Data Storage

Headlines are stored in `data/daily_pennsylvanian_headlines.json` with timestamps for historical tracking.

Future Improvements

Potential enhancements could include:

- Scraping multiple sports headlines per day
- Categorizing headlines by sport type
- Adding article summaries and links
- Implementing data analysis tools

Credits

Based on the basic-git-scraper-template by J. Lumbroso.

License

[Include your license information here]

Would you like me to expand on any section or add additional details?

Claude does not have the ability to run the code it generates yet. Claude does not have internet access. Links provided may not be accurate or up to date.

AG

give me the git code I can add directly to the file

Here's the markdown code you can directly copy and paste into your README.md file:

markdown

~~~
# Daily Pennsylvanian Sports Headline Scraper

A Python-based web scraper that automatically collects the top sports headline
daily from The Daily Pennsylvanian's sports section.

## Overview

This project modifies the [basic-git-scraper-template]
(https://github.com/jlumbroso/basic-git-scraper-template) to specifically track
sports coverage from Penn's student newspaper. While the original scraper
collected main headlines, this version focuses on sports content, providing a
way to monitor Penn's athletic coverage over time.

## Modifications Made

1. **URL Modification**:
   - Original: `https://www.thedp.com`
   - Modified: `https://www.thedp.com/section/sports`

2. **HTML Element Targeting**:
   - Original: `find("a", class_="frontpage-link")`
   - Modified: `find("h3", class_="standard_link")`

## Key Features

- Daily automated scraping of top sports headlines
- JSON-based storage for historical tracking
- Logging system for monitoring scraper performance
- Error handling for reliable operation

## Technical Implementation

The scraper uses:
- BeautifulSoup4 for HTML parsing
- Requests library for web scraping
- Loguru for comprehensive logging
- JSON for data storage

## Getting Started

1. Clone this repository
2. Install requirements:
   ```bash
   pip install -r requirements.txt
~~~

3. Run the scraper:

```
bash
```

```
python script.py
```

Data Storage

Headlines are stored in `data/daily_pennsylvanian_headlines.json` with timestamps for historical tracking.

Future Improvements

Potential enhancements could include:

- Scraping multiple sports headlines per day
- Categorizing headlines by sport type
- Adding article summaries and links
- Implementing data analysis tools

Credits

Based on the basic-git-scraper-template by J. Lumbroso.

```
You can directly copy this code into your README.md file. Note that I left out
the License section since you'll need to determine which license you want to
use for your project. Would you like me to suggest some common open-source
licenses you could use?
```

Claude does not have the ability to run the code it generates yet. Claude does not have internet access. Links provided may not be accurate or up to date.

Claude can make mistakes. Please double-check responses.