PAGE No.
DATE / /

My Name - Vaishali salve
Batch No - ds 2311
Date: 30/02/2024
Sir's - Shwetank Mishra

Assignment Name :- web-Scraping
Assignment 4

Q1 Scrape the details of most viewed videos on YouTube from wikipedia. url - https://en.wikipedia.org/wiki/list_of_most-viewed_youtube_videos you need to find following details: A) Rank, (B) name (C) Artist (D) upload date (E) views

⇒ To scrape the details of the most viewed videos on the YouTube from wikipedia you can use Python with libraries like Beautiful soup and requests. Here's a basic example of how you can achieve this:

Python

copy code

```
import requests
from bs4 import BeautifulSoup

# URL of the wikipedia page
url = "https://en.wikipedia.org/wiki/list_of_most-viewed_youtube_videos"
# send a GET request to the URL
response = requests.get(url)
```

```
# Parse the HTML content
soup = BeautifulSoup(response.text, 'html.parser')

# Find the table containing the most viewed videos.
table = soup.find('table', {'class': 'wikitable'})
# Initialize lists to store data
rank_list = []
name_list = []
artist_list = []
upload_date_list = []
views_list = []
# Iterate over each row in the table (skipping
                                    the header row)
for row in table.find_all('tr')[1:]:
# Extract data from each column in the row.
    columns = row.find_all('td')
    rank = columns[0].text.strip()
    name = columns[1].text.strip()
    artist = columns[2].text.strip()
    upload_date = columns[3].text.strip()
    views = columns[4].text.strip()
# Append data to respective lists.

    rank_list.append(rank)
    name_list.append(name)
    artist_list.append(artist)
    upload_date_list.append(upload_date)
    views_list.append(views)
```

# Print the scraped data

```
for i in range (len(Rank_list)):
print (f"Rank : {rank_list[i]}")
print (f"Name : {name_list[i]}")
print (f"Artist : {artist_list[i]}")
print (f"Upload Date : {upload_date_list[i]}")
print (f"Views : {views_list[i]}")
print ()
```

This script sends a GET request to the Wikipedia page, parses the HTML content using BeautifulSoup, finds the table containing the most viewed videos, extracts the required details (Rank, Name, Artist, Upload Date, Views) from each row, and then prints the scraped data. You might need to install the BeautifulSoup library if you haven't already (`pip install BeautifulSoup4`).

PAGE No.
DATE / /

Q2 Scrape the details team India's international fixtures from bcci.tv. url = https://www.bcci.tv/. you need to find following details: (A) series (B) place (C) date (D) time
note: from bcci.tv home page you have reach to the international fixture page through code.

=> To scrape the details of Team India's International fixtures from the Bcci website, you can use python with libraries like BeautifulSoup and requests. Here's how you can do it.

Copy code

```
python
import requests
from bs4 import BeautifulSoup
# URL of the BCCI website
url = "https://www.bcci.tv/"
# Send a GET request to the URL
response = requests.get(url)
# Parse the HTML content
soup = BeautifulSoup(response.text, 'html.parser')

# Find the link to the international fixtures page
fixtures_link = soup.find("a", text="International Fixtures")['href']
# Construct the URL for the international fixtures page
fixtures_url = url + fixtures_link
# Send a GET request to the international fixtures page
...
```

fixtures_response = requests.get(fixtures_url)

# Parse the HTML content of the fixtures page
fixtures_soup = BeautifulSoup(fixtures_response.text, 'html.parser')
# Find the section containing the fixtures
fixtures_section = fixtures_soup.find("ul", class_="js-list")
# Initialize lists to store data
series_list = []
place_list = []
date_list = []
time_list = []
# Iterate over each fixture
for fixture in fixtures_section.find_all("li"):

# Extract data from the fixture
series = fixture.find("p", class_="fixtures__additional-info").text.strip()
place = fixture.find("span", class_="u-dinskwebtext").text.strip()
date = fixture.find("span", class_="fixture__date").text.strip()
time = fixture.find("span", class_="fixture__time").text.strip()
# Append data to respective list

```
series_list.append(series)
place_list.append(place)
date_list.append(date)
time_list.append(time)

# print the scraped data
for i in range(len(series_list)):
    print(f"series: {series_list[i]}")
    print(f"place: {place_list[i]}")
    print(f"date: {date_list[i]}")
    print(f"Time: {time_list[i]}")
    print()
```

This script first navigates to the BCCI website, finds the link to the International fixtures page, constructs the URL for that page, sends a GET request to it and then parses the HTML content to extract the required details (series, place, date, time) for each fixture. Finally, it prints the scraped data.

PAGE No.
DATE / /

Q3 Scrape the details of state-wise GDP of India from statisticstimes.com.
URL = http://statisticstimes.com/
You have to find following details (A) Rank (B) State (C) GSDP (18-19) - at current prices (D) GSDP (19-20) - at current prices - (E) Share (18-19) (F) GDP (in billion)
Note: from statistictimes home page you have reach to company page through code.

⇒

Python ⊟ Copy code

```
import requests
from bs4 import BeautifulSoup
# URL of the statistics times website
url = "http://statisticstimes.com/"

# Send a GET request to the URL
response = requests.get(url)

# Parse the HTML content
soup = BeautifulSoup(response.text, 'html.parser')
# Find the link to the economy page
economy_link = soup.find("div", class_="dropdown-content").find("a", text="Economy")["href"]

# Construct the URL for the economy page
economy_url = url + economy_link
```

# Send a GET request to the economy page.
economy_response = requests.get(economy_url)

# Parse the HTML content of the economy page.
economy_soup = BeautifulSoup(economy_response.text, 'html.parser')

# Find the link to the Indian States GDP page.
indian_states_gdp_link = economy_soup.find("ul", class_="dropdown-menu").find('a', text="Indian States")['href']

# Construct the URL for the Indian States GDP page.
indian_states_gdp_response = requests.get(indian_states_gdp_url)

# Send a GET request to the Indian States GDP page.
indian_states_gdp_response = requests.get(indian_states_gdp_url)

# Parse the HTML content of the Indian States GDP page.
indian_states_gdp_soup = BeautifulSoup(indian_states_gdp_response.text, 'html.parser')

# Find the table containing the state-wise GDP details.
table = indian_states_gdp_soup.find("table", {"id": "table-id"})

# Initialize lists to store data.
rank_list = []
state_list = []
gsdp_18_19_current_prices_list = []
gsdp_19_20_current_prices_list = []
share_18_19_list = []
gdp_billion_list = []

PAGE No.
DATE / /

# Iterate over each row in the table (skipping the header row)
for row in table.find_all("tr")[1:]:
# Extract data from each column in the row
columns = row.find_all("td")
rank = columns[0].text.strip()
state = columns[1].text.strip()
gsdp_18_19_current_prices = columns[2].text.strip()
gsdp_19_20_current_prices = columns[3].text.strip()
share_18_19 = columns[4].text.strip()
gdp_billion = columns[5].text.strip()

# Append data to respective list
rank_list.append(rank)
state_list.append(state)
gsdp_18_19_current_prices_list.append(gsdp_18_19_current_prices)
gsdp_19_20_current_prices_list.append(gsdp_19_20_current_prices)
share_18_19_list.append(share_18_19)
gdp_billion_list.append(gdp_billion)
# print the scraped data
for i in range(len(rank_list)):
print("Rank: ", rank_list[i])
print("State: ", state_list[i])
print("GSDP (18-19)- current prices: ", gsdp_18_19_current_prices_list[i])
print("GSDP (19-20)- current prices: ", gsdp_19_20_current_prices_list[i])
print("Share (18-19): ", share_18_19_list[i])
print("GDP ($ billion): ", gdp_billion_list[i])

PAGE No.
DATE / /

This Scrapy first navigates to the statistics website, find the link to the economy page, constructs the URL that page, sends a GET request to it, find the link the India state GDP page, constructs the URL for that sends a GET request to it and then parse the HTML content to extract required details. (Rank, State, GSDP (18-19) - Current prices, GSDP (19-20) Current prices, Share (18-19), GDP $ billion) for each state. Finally, it print the scraped data

PAGE No.
DATE / /

Q4 Scrape the details of trending repositories of Github.com. URL - https://github.com/)

⇒ To Scrape the details of trending repositories you can use python with libraries like BeautifulSoup and requests Here's How you can do it

python. Copy code

```
import requests
from bs4 import BeautifulSoup
# Function to get the trending repositories data
def get_trending_repositories():
# URL of GitHub trending page.
    url = "https://github.com/trending"

# Send a GET request to the URL
response = requests.get(url)
# Parse the HTML content
soup = BeautifulSoup(response.text, 'html.parser')
# Find the list of trending repositories
repo_list = soup.find_all('article', class_='Box-row')

# Initialize lists to store data
titles = []
descriptions = []
contributors_counts = []
languages = []

# Extract data for each repository
for repo in repo_list:
```

```
# Extract repository title
title = repo.find("h1", class_="h3 lh-condensed").text.strip()
titles.append(title)
# Extract language used
language = repo.find("span", class_="d-inline-block mt-0 mr-3").text.strip()
languages.append(language)
# Return the scraped data
return title, descriptions, contributors_counts, languages

# Get the trending repositories data
title, descriptions, contributor_counts, language = get_trending_repositories()

# Print the scraped data
for i in range(len(titles)):
print(f"Repository Title: {title[i]}")
print(f"Repository Description: {description[i]}")
print(f"Contribution Count: {contribution_count[i]}")
print(f"Language Used: {languages[i]}")
print()
```

This script navigates to the GitHub trending page, parses the HTML content to extract details (Repository Title, Repository Description, Contribution Count, language used) for each trending repository, and then print the scraped data.

PAGE No.
DATE / /

Q5 Scrape the details of top 100 songs on Billboard.com. URL = http://www.billboard.com/
You have to find the following details:
(A) Song name (B) Artist name (3) Last week rank
(D) Peak rank (E) weeks on Board

Note: From the home page you have to click on the chart option then hot 100 page link through code

⇒

Python

```
import requests
from bs4 import BeautifulSoup
# Function to get the top 100 songs data
def get_top_100_songs():
# URL of Billboard's Hot 100 page
url = https://www.billboard.com/charts/hot-10

# Send a GET request to the URL
response = requests.get(url)
# Parse the HTML content
soup = BeautifulSoup(response.text, "html.parser")

# Find the list of top 100 songs
songs_list = soup.find_all("li", class_="chart-
                            list-element")
# Initialize lists to store data
song_names = []
artist_names = []
last_week_ranks = []
```

peak_ranks = []
weeks_on_board = []
# Extract song name
song_name = song.find('span', class_="chart-element__information__song").text.strip()
song_names.append(song_name)

# Extract artist name
artist_name = song.find('span', class_="chart-element__information__artist").text.strip()
artist_name.append(artist_name)

# Extract last week rank
last_week_rank = song.find('span', class_="chart element__meta text--center color--secondary text--last").text.strip()
#
Extract peak rank
peak_rank = song.find('span', class_="chart-element__meta text--center color--secondary text--peak").text.strip()
peak_ranks.append(peak_rank)

# Extract week on Board
weeks_on_board = song.find('span', class_="chart-element__meta text--center color--secondary text--week").text.strip()
week_on_board.append(weeks_on_board)

# Return the scraped data
return song_name, artist_name, last_week, ranks, peak_ranks, weeks_on_board

PAGE No.
DATE / /

# Get the top 100 songs data!
song_names, artist_names, last_week_ranks, peak_ranks, weeks_on_board = get_top_100_songs()

# Print the scraped data
for i in range(len(song_names)):
    print(f"song name: {song_names[i]}")
    print(f"artist name: {artist_names[i]}")
    print(f"last week Rank: {last_week_Rank[i]}")
    print(f"peak Rank: {peak_rank[i]}")
    print(f"weeks on Board: {weeks_on_board[i]}")
    print()

This script navigates to the billboard Hot 100 page parses the HTML content to extract details (song name, artist name, last week Rank, Peak, Rank week, weeks on Board) for each song in the top 100 list and then print the scraped data.

PAGE No.
DATE / /

Scrape the details of Highest selling novels.
(A) Book name (B) Author name (C) Volumes sold
(D) Publisher (E) Genre
url - https://www.theguardian.com/news/datablog/2012/aug/09/best-selling-book-all-time-fifty-shades-grey-compare

⇒ Python

Copy Code

```
import requests
from bs4 import BeautifulSoup
# Function to get the highest selling novels data
def get_highest_selling_novels():
    url = "https://www.theguardian.com/news/datablog/2012/aug/09/best-selling-book-all-time-fifty-shades-grey-compare"

# Send a GET request to the URL
response = requests.get(url)
# Parse the HTML content
soup = BeautifulSoup(response.text, 'html.parser')

# Find the table containing the highest selling novels data
table = soup.find('table', class_='in-article sortable')
# Initialize lists to store data
book_names = []
author_name = []
volumes_sold = []
publishes = []
genres = []
```

# Extract data for each novel
for row in table.find_all("tr")[1:3]:
columns = row.find_all("td")
book_name = columns[1].text.strip()
author_name = columns[2].text.strip()
volumns = columns[3].text.strip()
publisher = columns[4].text.strip()
genre = columns[5].text.strip()

# Append data to respective lists
book_names.append(book_name)
author_names.append(author_name)
volumes_sold.append(volumns)
publishers.append(publisher)
genres.append(genre)

# Return the scraped data
return book_name, author_names, volumes_sold, publishers, genres

# Get the highest seling novels data
book_name, author_name, volumes_sold, publisher, genres = get_highest_seling_novles()

# print the scraped data

PAGE No.
DATE / /

for i in range (len Book_name)):

print (f"Book name : {Book_names[i]}")
print (f"Author name : {author_names[i]}")
print (f"Volumes Sold : {volume_sold[i]}")
print (f"Publisher : {publishers[i]}")
print (f"Genre : {genres[i]}")
print ()

This script navigates to the guardian's webpage containing information about the highest selling novels, parses the HTML content extract details (Book name, Author name, Volume Sold, publisher, Genre) for each novel and then prints the scraped data.

Q7) Scrape the details most watched tv series of all time imdb.com url = https://www.imdb.com/list/ls095964455/ you have to find the following details
(A) name (B) year span (C) genre (D) run time
(E) Rating (F) votes

⇒

Python

Copy code

```
import request
from bs4 import BeautifulSoup

# function to get the most watched TV series data
def get_most_watched_tv_series():
# URL of the IMDb page containing the most watched
                                    TV series data
url = "https://www.imdb.com/list/ls095964455/"

# Send a GET request to the URL
response = requests.get(url)
# Parse the HTML content
soup = BeautifulSoup response.text, 'html.parser')
# Find the list of most watched TV series
series_list = soup.find_all('div', class_=
                                    'lister-item-content')
# Initialize lists to store data
names = []
year_spans = []
genre = []
run_time = []
rating = []
votes = []
```

```
# Extract data for each TV series
for series in series_list:
    # Extract series name
    name = series.find("h3").a.text.strip()
    names.append(name)
    # Extract year span
    year_span = series.find("span", class_="lister-item-year
        text-muted unbold").text.strip()
    year_span.append(year_span)
    # Extract genre
    genre = series.find("span", class_="genre").text.strip()
    genres.append(genre)

    # Extract run time
    run_time = series.find("span", class_="Runtime").text.strip()
    run_times.append(run_time)
    # Extract rating
    rating = series.find("span", class_="ipl-rating-star
        rating").text.strip()
    ratings.append(rating)
    # Extract votes
    vote = series.find("span", attrs={"name": "nv"}
        ).text.strip()
    votes.append(vote)
# Return the scraped data

return names, years, spans, genres, run_
times, ratings, votes.
```

PAGE No.
DATE / /

```
# Get the most watched TV series data
names, years, spans, genres, run_times,
rating, votes, = get_most_watched_tv_
series()

# Print the scraped data
for i in range(len(names)):

print(f"Name: {name[i]}")
print(f"Year Span: {years_spans[i]}")
print(f"Genre: {genres[i]}")
print(f"Run time: {runtimes[i]}")
print(f"Ratings: {ratings[i]}")
print(f"Votes: {votes[i]}")
print()
```

This script navigates to the IMDb page containing information about the most watched TV series, parses the HTML content to extract data (Names, Years, Span, Genre, Run time, Rating, Votes) for each series, and prints the scraped data.

Q8. Details of Datasets from UCL machine learning repositories. URL = http://archive.ics.edu/ you have to find the following details:
(A) Dataset name (B) Data type (C) Task
(D) Atribute type (E) no of instance (G) year

=> Python

```
import request
from bs4 import BeautifulSoup
# function to get the dataset details
def ge-dataset-details():
    # URL of the UCI machine learning Repository homepage
    url = "http://archive.ics.uci.edu/"

    # Send a GET request to the URL
    response = request.get(url)
    # Parse the HTML content
    Soup = BeautifulSoup(response.text html.parser)

    # Find the link to the "show all Dataset" page
    all_dataset_link = soup.find("a", href= dataset.sp)
    # Construct the URL for the show All Dataset "page
    all-dataset-url = url + all-dataset-link["href"]

    # Send a GET request to the "show" All Dataset "page
    all dataset-respon = requests.get(all-dataset-url)
    # Parse the HTML content of the show All
    Dataset" page
    all-dataset-Soup = BeautifulSoup(all-dataset-
    response.text, html.parser)
```

PAGE No.
DATE / /

# find the table containing the dataset details
table = cell_dataset.soup.find("table", class_="table")

# Initialize list to store data
dataset_names = []
data_types = []
tasks = []
attribute_types = []
num_attributes = []
years = []

# Extract data for each dataset
for row in table.find_all('tr')[1:]:
columns = row.find_all("td")
dataset_name = columns[0].text.strip()
data_type = columns[1].text.strip()
task = columns[2].text.strip()
attribute_type = columns[3].text.strip()
num_instance = columns[4].text.strip()
num_attribute = columns[5].text.strip()
year = colums[6].text.strip()

# Append data to respective lists
dataset_names.append(dataset_name)
data_types.append(data_type)
tasks.append(task)
attribute_types.append(attribute_type)
num_instances.append(num_instance)
num_attributes.append(num_attribute)
years.append(year)

# Return the scraped data,
return dataset_names, data_types, tasks, attribute_types, num_instances, num_attributes, years

# Get the dataset details
dataset_names, data_types, tasks, attribute_types, num_instances, num_attributes, years = get_dataset_details()

# print the dataset details
for i in range(len(dataset_names)):
print(f"Dataset Name: {dataset_name[i]}")
print(f"Data Type: {data_types[i]}")
print(f"Task: {tasks[i]}")
print(f"Attribute Types: {attribute_types[i]}")
print(f"No of Instances: {num_instances[i]}")
print(f"No of Attributes: {num_attributes[i]}")
print(f"year: {years[i]}")
print()

This script navigates to the UCI machine learning Repository homepage, find the link to the "Show All Dataset" page, constructs the URL for the next page, sends a GET request to it and then parse the HTML content to extract details (Dataset name, Data task, Attribute Type, No. of instances, No. of Attributes, year) for each dataset. Finally it print the scraped data.