UNIT - II

📓 **Topic: Web Crawling Models and Planning Web Scraping**

□ 1. Web Crawling Models

Definition:
Web crawling is the process of automatically navigating and collecting data from websites using web crawlers or spiders. These are programs designed to browse the internet and extract structured data.

Key Points:

- Crawlers can gather data such as news articles, product prices, customer reviews, etc.
- They use HTML tags like <h1> to extract titles and other elements.
- Flexibility is needed to:
  - Select websites to scrape.
  - Modify extraction logic without writing multiple lines of code.
- Can help compare product prices across sites and in various currencies.

Applications:

- Collecting and comparing product data.
- Extracting news articles or blog posts.
- Monitoring changes over time.

📝 2. Planning and Defining Objects

Definition:
This step involves identifying what data to extract and how to organize it. Proper planning helps avoid collecting unnecessary or unstructured data.

Common Data Fields for a Product (Example: Clothing Store):

- Product name
- Price
- Description
- Sizes
- Colors
- Fabric type
- Customer rating
- SKU (Stock Keeping Unit)
- Customer reviews

Important Considerations:

- Avoid adding unnecessary fields that increase complexity.
- Ensure data fields are reusable across multiple websites.
- Check if fields are specific to certain websites or general enough for all.

🔍 3. Designing Data Structure

Checklist Questions Before Finalizing Data Fields:

- Will this field help meet project goals?
- Is the field "nice to have" or essential?
- Is the data already available elsewhere?
- Will it be difficult to collect or maintain?

Product Object Example Fields:

- Product title
- Manufacturer
- Product ID (if available)
- Attributes (name, value pairs like "Color: Red")

Flexible Design Tip:
Use JSON format to store variable attributes for flexibility and easier data management.

🔄 4. Handling Product Variants and Prices

When tracking prices over time, collect:

- Store ID
- Product ID
- Price
- Date/Timestamp

Handling Product Variants (e.g., different skirt sizes):

- Use instance-level data:
    - Product ID
    - Instance type (e.g., size)
    - Instance ID
    - Price

- Date/Timestamp

📰 **5. Extracting Article Data**

Common Fields for News Articles:

- Title
- Author
- Date
- Content

Optional Fields (based on project):

- Revision dates
- Related articles
- Number of social shares

✨ **Conclusion:**

Before diving into writing web scraping code:

- Plan what data to collect.
- Understand the structure of the websites.
- Design flexible and reusable data models.
- Prioritize project-relevant fields.

✅ **Pro Tip for Exams:** Use diagrams to show a sample data schema or flow of a web crawler to score better in 10-mark questions.

Web Scraping – Dealing with Different Website Layouts

1. Introduction to Web Scraping (Definition)

Web scraping is the process of extracting data from websites. The data can be extracted from HTML pages and saved for further analysis or processing. Web scraping is useful in gathering content like news articles, product information, or prices from different sites.

---

2. Problem with Different Website Layouts

UNIT - II

Websites have various structures, making it difficult to use one solution for all. Each website can have different tags, layouts, or HTML elements for titles, bodies, and content sections.

---

3. General Approach to Handle Different Layouts

To deal with layout differences, one can:

- Create **separate web crawlers or parsers** for each website.
- Use **CSS selectors** and **BeautifulSoup** to extract the correct content.
- Use a class-based structure to organize the scraping logic.

---

4. Content Class Example

The Content class represents extracted web content from a single web page:

```
class Content:
    def __init__(self, url, title, body):
        self.url = url
        self.title = title
        self.body = body
```

This class stores the:

- **URL** of the content
- **Title** of the article
- **Body** or main content

---

5. Web Scraper Function Example

A sample function using BeautifulSoup:

```
def scrapeNYTimes(url):
    bs = getPage(url)
    title = bs.find('h1').text
    lines = bs.select('div.StoryBodyCompanionColumn div p')
    body = '\n'.join([line.text for line in lines])
    return Content(url, title, body)
```

This function:

- Parses the webpage using BeautifulSoup

- Extracts the **title**
- Extracts the **body text** using CSS selectors

---

6. Website Class

This class stores metadata about the website structure:

```
class Website:
    def __init__(self, name, url, titleTag, bodyTag):
        self.name = name
        self.url = url
        self.titleTag = titleTag
        self.bodyTag = bodyTag
```

It doesn't store the content itself, but **guides the scraper** on how to extract title and body using CSS selectors.

---

7. Crawler Class and Dynamic Parsing

The Crawler class is responsible for:

- Accessing pages
- Using Website metadata to know which tag to look for
- Creating Content objects dynamically for different sites

```
def safeGet(self, pageObj, selector):
    selectedElems = pageObj.select(selector)
    return '\n'.join([elem.get_text() for elem in selectedElems]) if selectedElems else ''
```

---

8. Advantages of this Approach

- Easy to extend to multiple sites
- Clean structure using classes
- Reduces code duplication
- Easier for non-programmers to update parsing rules

---

9. Disadvantages

- Less flexible than writing custom logic for each site

- Sites must follow the expected structure

---

10. Conclusion

Handling different website layouts in web scraping requires flexible design using object-oriented programming. Using classes like Content, Website, and Crawler, scraping logic becomes modular, scalable, and easier to maintain.

Web Scraping & Scrapy – (10 Marks Answer)

1. Structuring Crawlers

Definition:

Crawlers (or spiders) are automated programs that systematically browse websites and extract information. Structuring crawlers involves designing them to handle different types of websites and extract the required data efficiently.

Key Points:

- **Reusability:** Creating flexible and reusable crawlers avoids writing new ones for each website.
- **Search Pages:** Many websites use search pages that return lists of results using URL query parameters (e.g., example.com/Search?topic=Python).
- **Link Identification:** Resulting pages have links (internal or external) that are identified using CSS classes, HTML tags, etc.
- **Normalization:** After extracting the result links, URLs need to be normalized (absolute/relative).
- **Main Goal:** Reduce the crawling problem to structured data extraction from a web page.

---

2. Scrapy Framework

Definition:

Scrapy is an open-source Python framework used for web scraping. It extracts data from websites and processes it to be stored in formats like JSON, CSV, or databases.

Key Features of Scrapy:

Feature	Description
Fast	Built on Twisted (asynchronous networking library)
Extensible	Customizable with middleware and pipelines
Exporting Options	JSON, CSV, XML, databases

Feature	Description
HTML Parsing	Handles HTML, XPath, CSS selectors
Usability	Suitable for small/large-scale projects

---

3. Scrapy Project Structure

When you create a Scrapy project using the command scrapy startproject mycrawler, the folder structure looks like:

```
mycrawler/
|
├── mycrawler/           # Main Python module
|   ├── items.py         # Define data structures
|   ├── middlewares.py   # Middleware customization
|   ├── pipelines.py     # Data processing
|   ├── settings.py      # Project settings
|
├── spiders/             # Spider definitions
|   └── example_spider.py  # Example spider
|
└── scrapy.cfg           # Project configuration
```

---

4. Example Scrapy Spider Code

```
import scrapy


class ExampleSpider(scrapy.Spider):
    name = "example"
    start_urls = ['https://quotes.toscrape.com/']


    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('span small::text').get(),
                'tags': quote.css('div.tags a.tag::text').getall(),
            }

        next_page = response.css('li.next a::attr(href)').get()
        if next_page:
            yield response.follow(next_page, self.parse)
```

---

5. Advantages of Scrapy:

- Fast and asynchronous
- Easy to use CSS/XPath selectors
- Supports pagination
- Export data in multiple formats
- Scalable for large scraping tasks

Conclusion:

Scrapy is a powerful Python framework designed for building web crawlers and scrapers. With its modular architecture and strong support for HTML parsing and data exporting, it makes data extraction efficient and reliable.

Web Scraping Using Scrapy – 10 Marks

---

1. Structuring Crawlers

Definition:
Crawlers (or spiders) are scripts used to extract structured data from websites. They follow links, download pages, and extract information.

Key Points:

- Most search engines use URLs with parameters to search topics.
- Scrapers extract lists and details by analyzing HTML structure.
- Scrapy simplifies the crawler design by providing reusable methods.

UNIT - II

2. What is Scrapy?

Definition:
Scrapy is an open-source Python framework used for web scraping and extracting structured data from websites, then saving it in formats like JSON, CSV, or databases.

Key Features of Scrapy:

- Built on Twisted (asynchronous networking).
- Modular and extensible.
- Supports crawling, scraping, storing data (JSON, XML, CSV).
- Easy integration with pipelines and middleware.

---

3. Installing Scrapy

Steps:

- Prerequisites: Python 3.6+, pip, virtualenv.
- Traditional installation:
- pip install scrapy
- Virtual environment (recommended):
- python -m venv scrapy_env
- scrapy_env\Scripts\activate
- pip install scrapy

---

4. Creating a Scrapy Project

Command:

scrapy startproject mycrawler

Project Structure:

```
mycrawler/
|
├── mycrawler/
|   ├── __init__.py
|   ├── items.py
|   ├── middlewares.py
|   ├── pipelines.py
|   ├── settings.py
|  └── spiders/
|      └── example.py
└── scrapy.cfg
```

---

5. Writing a Simple Spider

Command to generate spider:

```
scrapy genspider example quotes.toscrape.com
```

Sample Spider Code:

```
class ExampleSpider(scrapy.Spider):
    name = 'example'
    start_urls = ['https://quotes.toscrape.com/']

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('span small::text').get(),
                'tags': quote.css('div.tags a.tag::text').getall(),
            }
```

---

6. Spidering with Rules

Definition:
Used for crawling multiple pages using rules and link extractors (e.g., CrawlSpider, Rule, LinkExtractor).

Purpose: Automatically follows links and processes them using defined callback functions.

---

7. The Item Pipeline

Definition:
Pipelines process scraped data after extraction. They allow cleaning, validation, and saving of data to storage systems.

Example:

```
ITEM_PIPELINES = {
    'wikiSpider.pipelines.WikispiderPipeline': 300,
}
```

---

8. Logging with Scrapy

Definition:
Scrapy has a logging system to monitor spider activity.

Logging Levels:

- CRITICAL
- ERROR
- WARNING
- DEBUG
- INFO

Example:

```
LOG_LEVEL = 'ERROR'
```

---

9. Storing Data

Scrapy allows exporting data to:

- JSON
- CSV
- XML

Command:

```
scrapy crawl example -o quotes.json
```

---

10. Media Files Handling

Two methods:

- By reference (store URLs)
- By downloading files

Advantages of reference:

- Saves space and bandwidth.
- Less load on target server.

UNIT - II

Conclusion

Scrapy is a powerful, fast, and efficient framework for building web crawlers and scrapers. It supports complex tasks like link following, data processing, and saving outputs in multiple formats, making it ideal for large-scale data extraction projects.

Topic: MySQL and Integrating MySQL with Python (Web Scraping Context)

---

1. What is MySQL?

- **Definition:**
  MySQL is an **open-source relational database management system (RDBMS)**. It stores data in tables and is known for being robust, scalable, and efficient.
- **Pronunciation:**
  Officially pronounced **"My Es-Kew-El"**, though many say **"My Sequel"**.
- **Popularity:**
  It's one of the **most widely used databases**, often compared with **Oracle DBMS** and **Microsoft SQL Server**.
- **Used by Major Companies:**
  Websites like **YouTube, Twitter, Facebook** use MySQL due to its reliability.

---

2. MySQL Installation Overview

- **For Windows:**
  A user-friendly **installation wizard** is available.
- **For macOS:**
  Use **Homebrew** package manager:
- $ brew install mysql
- $ cd /usr/local/mysql
- $ sudo ./bin/mysqld_safe

---

3. Integrating MySQL with Python

- **Why integrate?**
  Python doesn't have built-in support for MySQL, so we use **external libraries** like **PyMySQL**.
- **Installing PyMySQL:**
- pip install PyMySQL
- **Sample Python Code for Connecting:**

- import pymysql
- conn = pymysql.connect(host='127.0.0.1', unix_socket='/tmp/mysql.sock', user='root',
-                   passwd=None, db='scraping')
- cur = conn.cursor()
- cur.execute("USE scraping")
- **Closing Connection:**
  Always close connections to avoid memory leaks:
- cur.close()
- conn.close()

---

4. Unicode Support in MySQL

- **Problem:**
  MySQL's default encoding (utf8mb4) lacks complete Unicode support.
- **Solution:**
  Set character set to **utf8mb4_unicode_ci** for better Unicode handling:
- ALTER DATABASE scraping CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci;
- ALTER TABLE pages CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

---

5. Storing Scraped Data

- **Required Libraries:**
- from urllib.request import urlopen
- from bs4 import BeautifulSoup
- import pymysql, datetime, random, re
- **Storing Data in Database:**
- def store(title, content):
-     cur.execute("INSERT INTO pages (title, content) VALUES (%s, %s)", (title, content))
-     conn.commit()

---

6. Extracting Data from Wikipedia

- **Example Code to Scrape Title:**
- def getLinks(articleUrl):
-     html = urlopen("http://en.wikipedia.org" + articleUrl)

- bs = BeautifulSoup(html, 'html.parser')
- title = bs.find('h1').get_text()

Summary for Exam Writing (10 Marks):

➤ **Introduction (2 Marks):**

- Define MySQL.
- Mention its popularity and usage in web applications.

➤ **Installation (1 Mark):**

- Simple mention of how to install on Windows/macOS (Homebrew).

➤ **Integration with Python (2 Marks):**

- Using PyMySQL for database access.
- Installation via pip.

➤ **Sample Python Code (2 Marks):**

- Connecting to MySQL using Python.
- Writing and reading data using cursor().

➤ **Handling Unicode (1 Mark):**

- Changing character set to support non-ASCII data.

➤ **Web Scraping Integration (2 Marks):**

- Using BeautifulSoup and urllib to extract data.
- Storing scraped data into MySQL.

---

1. Web Crawling and Scrapy (Python + MySQL Integration)

Definition:

Web crawling is an automated method to systematically browse and extract content from web pages. Python libraries like BeautifulSoup, urllib, and database connectors like PyMySQL are commonly used.

Important Concepts:

- **BeautifulSoup**: Used for parsing HTML content.
- **Random Selection**: random.randint() is used to pick random links.
- **MySQL Connection**: Connect to local MySQL using pymysql.connect().

- **Store Data**: Pages and links are inserted into MySQL tables using INSERT INTO.

---

2. Database Techniques and Good Practices

Definition:

Database techniques are structured approaches for organizing and managing data effectively in relational databases like MySQL.

Key Points:

- **Use ID Columns**: Always include a primary key (id) that is AUTO_INCREMENT.
- **Meaningful Names**: Use descriptive table and column names for readability.
- **Intelligent Indexing**: Use indexed columns for faster searches.
- **Normalization**:
    - **OLTP**: Normalize to avoid redundancy.
    - **OLAP**: Denormalize for performance in analytics.
- **Use Foreign Keys**: For referential integrity between tables.
- **Optimize Queries**: Use EXPLAIN, ANALYZE to improve performance.
- **Backup Frequently**: Always test restore capabilities.

---

3. Six Degrees in MySQL

Definition:

Based on the “Six Degrees of Kevin Bacon” concept, the goal is to find relationships between Wikipedia pages using link chains.

Concept Overview:

- **Page and Link Storage**: Store pages and the connections (links) between them.
- **Use Case**: Demonstrates relational mapping using pageId, toPageId in MySQL.
- **SQL Example**:
- INSERT INTO links (fromPageId, toPageId) VALUES (A, B);

---

4. Email Integration using Python

Definition:

Python can be used to send emails via the SMTP protocol using smtplib and email.mime.

Key Points:

- **Modules Used**: smtplib, email.mime.text.MIMEText

- **SMTP Connection**: Connect to localhost SMTP server.
- **Code Structure**:
- from email.mime.text import MIMEText
- msg = MIMEText("body")
- msg['Subject'] = "Alert"
- msg['From'] = "sender@example.com"
- msg['To'] = "receiver@example.com"
- s = smtplib.SMTP('localhost')
- s.send_message(msg)
- s.quit()

---

5. Python Functions with MySQL

Functions Defined:

- **insertPageIfNotExists**: Inserts a page if it doesn’t already exist.
- **insertLink**: Inserts a link only if it’s not already present.
- **loadPages**: Loads current pages from DB for crawling.

Sample Tables (MySQL):

```
CREATE TABLE pages (
  id INT NOT NULL AUTO_INCREMENT,
  url VARCHAR(255) NOT NULL,
  created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (id)
);


CREATE TABLE links (
  id INT NOT NULL AUTO_INCREMENT,
  fromPageId INT NOT NULL,
  toPageId INT NOT NULL,
  created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (id)
```

);

Tips for Writing in Exams (10 Marks Format):

- Start with definitions.
- Use bullet points for best practices.
- Include **code snippets** or **SQL statements** where relevant.
- Summarize each section clearly with 2–3 key sentences.