Web Crawler Overview

Got it — you want a **straight, no-nonsense, exam-ready explanation.**
I'll cover everything from your notes **clearly**, **simply**, and **completely** without missing anything. Let's go.

🚀 Full Exam-Ready Explanation

📚 Search Engine

- A **search engine** is a program that **helps you find information** on the Internet.
- It takes your **query**, searches its **index** (a database of pages), and shows **relevant** results.

How it works (Important)

1. **Crawling**: Collect web pages.
2. **Indexing**: Organize and store information from those pages.
3. **Searching**: Match your query to indexed pages.
4. **Ranking**: Sort results by relevance and quality.
5. **Displaying**: Show titles, links, and snippets.

🕷️ Web Crawler

- A **web crawler** (also called **spider** or **bot**) is a **program that automatically visits websites** to **find and collect** information.
- It starts with some known URLs (called **seed URLs**), fetches the pages, **extracts links**, and **keeps visiting** new links.

Steps:

1. Start with **seed URLs**.

2. Visit each URL (send HTTP request).
3. Collect page data (text, images, links, metadata).
4. Extract and add new links.
5. Repeat till all pages are visited or a limit is reached.

📑 Indexing

- After crawling, **indexing** makes the data searchable.

Steps:

1. **Extract Text** (ignore HTML tags).
2. **Tokenization**: Split into words.
3. **Stop-word Removal**: Remove common words ("the", "is", etc.).
4. **Stemming/Lemmatization**: Reduce words to their base form.
5. **Inverted Index**: Store a map of words → page locations.
6. **Metadata Storage**: Save titles, keywords, etc.

✅ Example of Inverted Index:

- Word "learning" → Page1, Page2, Page3

🥊 Web Crawling vs Web Scraping

Aspect	Web Crawling	Web Scraping
Goal	Visit and index the **whole website**.	Extract **specific information**.
Usage	Search engines (Google, Bing).	Data analysis, market research.
Technique	Follow all links automatically.	Target specific pages/fields manually.

🛠️ Applications of Web Crawling

- **Search Engine Indexing** (Googlebot, Bingbot, etc.)
- **Website Optimization** (SEO improvements)
- **Market Research** (competitor data)
- **Social Media Monitoring** (user activity, sentiment)
- **News and Media Tracking** (latest events)
- **E-commerce** (price tracking)
- **IP Protection** (detect copyright violations)

🌟 Examples of Popular Web Crawlers

- **Googlebot** (Google)
- **Bingbot** (Microsoft Bing)
- **DuckDuckBot** (DuckDuckGo)
- **Amazonbot** (Amazon)
- **Yahoo Slurp** (Yahoo)
- **Yandex Bot** (Yandex)

🚧 Challenges in Web Crawling

- **Server load**: Can overload websites.
- **Bandwidth consumption**: Can waste resources.
- **Privacy concerns**: Risk of collecting sensitive info.
- **Copyright issues**: Copying protected content.
- **Duplicate content**: Hard to handle repeated pages.
- **Legal issues**: Must respect different countries' laws.

🛠️ Writing Your First Simple Crawler (with Explanation)

```python
import requests
from bs4 import BeautifulSoup

def simple_crawler(url):
    response = requests.get(url)  # Fetch page
    soup = BeautifulSoup(response.text, 'html.parser')  # Parse page
    links = set()
    for link in soup.find_all('a'):
        links.add(link.get('href'))  # Collect all href links
    print(f"Found {len(links)} links:")
    for link in links:
        print(link)

simple_crawler('https://www.geeksforgeeks.org/')
```

🔥 What this code does:

- Requests the page.
- Parses HTML.
- Extracts all `<a>` tag links (`href`).
- Prints all unique links.

🕸️ Scrapy Framework

- **Scrapy** is a **Python framework** for **fast** and **efficient** web crawling and scraping.
- It's much more **powerful** than basic BeautifulSoup.

🏛️ Scrapy Architecture Components:

Component	Role
Scrapy Engine	Controls data flow.
Scheduler	Queues URLs to be visited.
Downloader	Downloads web pages.
Spiders	Extract info from pages (your custom logic).
Item Pipeline	Cleans, validates, and stores scraped data.
Downloader Middleware	Hooks between Downloader and Engine (can modify requests).
Spider Middleware	Hooks between Spider and Engine (can modify responses/requests).

🔄 Data Flow in Scrapy

1. **Spider** sends **Requests** → **Engine** → **Scheduler**.
2. **Scheduler** returns next **Request** → **Engine** → **Downloader**.
3. **Downloader** fetches the page → **Engine** → **Spider**.
4. **Spider** extracts data and/or new Requests → **Engine**.
5. **Item Pipeline** processes extracted items.
6. Repeat till no more Requests.

🔥 Example: Scraping with Scrapy Selector

```python
from scrapy.selector import Selector

html = """
<html>
<body>
<h1>Welcome to My Site</h1>
```

```
<p>This is a sample paragraph.</p>
<a href="https://example.com">Visit Example</a>
<img src="https://www.w3schools.com/html/pic_trulli.jpg" alt="Example Image">
</body>
</html>
"""

selector = Selector(text=html)
heading = selector.xpath('//h1/text()').get()
paragraph = selector.css('p::text').get()
link_text = selector.css('a::text').get()
link_href = selector.css('a::attr(href)').get()
image_src = selector.css('img::attr(src)').get()

print("Heading:", heading)
print("Paragraph:", paragraph)
print("Link Text:", link_text)
print("Link Href:", link_href)
print("Image Src:", image_src)
```

🛒 Scraping multiple elements

python

```python
html = """
<html>
<body>
<div class="product">
<h2 class="title">Bat</h2>
<span class="price">$10</span>
</div>
<div class="product">
<h2 class="title">Ball</h2>
<span class="price">$20</span>
</div>
</body>
</html>
"""
```

```
selector = Selector(text=html)
for product in selector.css('div.product'):
    print(product.css('h2.title::text').get())
    print(product.css('span.price::text').get())
```

🕵️ Full Scrapy Crawler Example

```python
import scrapy
from scrapy.crawler import CrawlerProcess

class QuotesSpider(scrapy.Spider):
    name = "quotes"

    def start_requests(self):
        yield scrapy.Request(url='http://quotes.toscrape.com', callback=self.parse)

    def parse(self, response):
        for quote in response.css('div.quote'):
            text = quote.css('span.text::text').get()
            author = quote.css('small.author::text').get()
            tags = quote.css('div.tags a.tag::text').getall()
            print(f"Quote: {text}")
            print(f"Author: {author}")
            print(f"Tags: {', '.join(tags)}")

# Run the spider
process = CrawlerProcess()
process.crawl(QuotesSpider)
process.start()
```

🖥️ Scrapy Shell

- **Scrapy Shell** = A command-line environment to:

  - Fetch a webpage
  - Try out CSS/XPath queries interactively

How to open:

```bash
scrapy shell http://quotes.toscrape.com
```

- You can see HTML using `print(response.text)`.
- Extract content using:

```python
response.css('span.text::text').get()
response.css('small.author::text').get()
response.css('div.tags a.tag::text').getall()
```

📱 Social Media Mining

- **Social media mining** = Getting useful **patterns, insights** from social media (Twitter, Facebook, Reddit, etc).
- Uses **Data Mining, Machine Learning, NLP, Network Analysis**.

Applications:

- Sentiment analysis (happy/sad posts)
- Trend detection
- Influencer identification
- Crisis monitoring
- Market research

📥 Data Collection

- Two methods:
  - **Web Scraping**: Scrape posts manually.
  - **APIs**: Use provided official APIs (Twitter API, Facebook API, etc).

What data we collect:

- Posts/tweets
- Likes, shares, comments
- User profiles
- Hashtags, mentions
- Timestamps, locations

🧹 Data Preprocessing

- Remove junk: URLs, mentions (@user), hashtags (#topic).
- **Tokenization**: Split into words.
- **Stop-word removal**: Remove boring common words.
- **Stemming/Lemmatization**: Reduce words to their root form.

📦 Data Extraction

- Extract **important info** like:
  - Text, metadata
  - Entities (people, places)
  - POS tagging

📊 Exploratory Data Analysis (EDA)

- Understand data by:

    - Word frequency
    - Sentiment scores
    - Hashtag trends
    - Most active users/places

🗺️ Geo Visualization

- Mapping location-based data like **geotagged tweets**.
- Used for:
    - Crisis mapping
    - Regional sentiment analysis
    - Event detection
- **Tools**: GIS (ArcGIS, QGIS), Python (folium, geopandas, basemap).

🏆 Real Use Cases of Social Media Mining

Example	Use
Coca-Cola	Sentiment analysis to check public reaction.
United Airlines	Crisis management after PR disaster.
CDC (Flu tracking)	Monitor flu outbreaks using tweets.
LEGO Ideas	Get product feedback from community posts.

✅ Done.

No skipping. No sugar-coating. Full exam coverage in simple words.

Would you also like me to give you a **few rapid-fire sample MCQs or one-word questions** to revise this faster? 🎯
(Just say if you want.)

Straight up: **No**, you don't need to learn *all* the other codes deeply for a **theory exam**, especially if your exam mainly focuses on concepts.

Here's the deal:

Code	Necessary to learn by heart?	Why/When?
First Simple Crawler (`requests + BeautifulSoup`)	✅	Must know — It's basic and easy to explain if asked how crawling works practically.
Scrapy HTML Data Extraction (Selector usage)	⚠️ (Know basics, not full code)	Just understand *how selectors (CSS/XPath)* are used to pick elements. Don't memorize every line.
Scrapy Full Spider Example (`QuotesSpider`)	❌ (Theory enough)	You should know *what a Spider does*, *start_requests*, *parse method*. You don't have to memorize full code unless asked to implement.
Scrapy Shell usage	⚠️ (Just commands)	Know **how to open Scrapy shell** and **how to extract using selectors**. No need for deep code writing.
Social Media Mining code parts	❌	Mostly conceptual for exam: focus on steps like *data collection*, *preprocessing*, *geo-visualization*. No heavy code expected here.

So for theory exams, focus on:

- Concepts (how crawlers work, how scraping differs from crawling, how Scrapy architecture flows, etc.)
- Basic code ideas (not full implementations unless your syllabus *specifically* says *"coding questions"*)
- Simple commands (like `scrapy shell`, `print(response.css('selector'))` etc.)

Summary for you:
✅ Fully understand and be ready to explain **Simple Crawler** code.
⚡ Just know what **Scrapy** is and what its parts (Spiders, Scheduler, Downloader, Pipelines) do.
✅ Know basics of **Scrapy selectors** (CSS and XPath usage examples).
❌ You don't need to mug up full **Scrapy spider programs** or **social media mining codes**.

If you want, I can even give you a small **one-page cheat sheet** you can glance at before the exam. Want that? 📄💨
(Will save you last-minute panic.)

Good call. Here's a **tight, exam-focused cheat sheet** for you:
(Just the essentials — no fluff.)

📜 Web Crawling & Scraping Cheat Sheet (Theory Exam Focus)

---

1. First Simple Crawler (requests + BeautifulSoup)

Main Steps:

```python
import requests
from bs4 import BeautifulSoup

url = 'https://example.com'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
```

```
for link in soup.find_all('a'):
    print(link.get('href'))
```

Key Points:

- `requests.get(url)` fetches page content.
- `BeautifulSoup(response.text, 'html.parser')` parses the HTML.
- `find_all('a')` finds all anchor (`<a>`) tags — basically all hyperlinks.

In simple words:

> Request → Parse → Extract.

---

2. Scrapy Basics

Scrapy Key Components:

- **Spider**: Class where you write how to crawl (URLs) and what to extract.
- **Scheduler**: Decides which URL to visit next.
- **Downloader**: Downloads page content.
- **Item Pipeline**: Cleans and saves scraped data (DB, CSV etc).

Simple Scrapy Spider (flow):

```python
import scrapy

class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = ['http://quotes.toscrape.com/']

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {'text': quote.css('span.text::text').get()}
```

Key Points:

- `start_urls`: Starting page(s).
- `parse()`: Main method where extraction happens.
- `css('div.quote')`: Selects all `<div class="quote">` elements.

3. Scrapy Shell Commands

Quick examples:

bash

```bash
scrapy shell 'http://quotes.toscrape.com/'
```

Then in shell:

python

```python
response.css('title::text').get()
response.xpath('//title/text()').get()
```

- `css('selector')`: Select with CSS-style.
- `xpath('path')`: Select with XPath.

4. Crawling vs Scraping

Crawling	Scraping
Discovering links/pages	Extracting specific data
Focus: "Where to go next"	Focus: "What data to take"
Example: Googlebot	Example: Scrapy spider

5. Ethical Guidelines

- Always check `robots.txt`.
- Don't overload servers.
- Respect site policies (no stealing sensitive/private data).

🎯 How to Answer Theory Questions

If asked:

- *"What is a crawler?"* → "A program that automatically browses websites and collects links for crawling or data extraction."
- *"What is Scrapy?"* → "An open-source web crawling framework written in Python used to build spiders to scrape structured data."
- *"Difference between requests + BeautifulSoup vs Scrapy?"* → "Requests + BS4 is for simple one-off scraping. Scrapy is more powerful — manages crawling large sites efficiently."

📈 Final Tip

If stuck, always fall back to:

> "Crawling is about discovering. Scraping is about extracting."

Done.
This is *all* you need to walk into your theory exam feeling like a boss. 😎

If you want, I can also quickly throw in **2-3 sample theory questions with model answers** you can practice now. Want that too? 📚