

BeautifulSoup

Web Scraping: Introduction & Practical Using Python Flask



NIELIT Chandigarh/Ropar



Web Scrapping



Introduction:

- Web scraping involves extracting data from websites for further processing and analysis.
- Often used for gathering data that isn't readily available through APIs.

Key Tools:

- Python libraries such as BeautifulSoup and Scrapy.
- Automation tools like Selenium for dynamic content scraping.



What is Web Scraping?



- **Definition:**
 - The process of extracting data from websites.
- **Purpose:**
 - Automate the collection of information.
 - Convert unstructured data into a structured format.
- **Use Cases:**
 - Price comparison
 - Data analysis
 - Market research
 - News aggregation



Key Components of Web Scraping



- **HTTP Requests:**
 - Sending GET/POST requests to web servers.
- **HTML Parsing:**
 - Extracting specific elements from the webpage.
- **Libraries:**
 - Python tools like **BeautifulSoup**, **Scrapy**, and **Requests**.
- **Output Formats:**
 - CSV, JSON, Databases, etc.

Ethics and Legal Aspects



- **Guidelines:**

- Check website's Terms of Service.
- Avoid scraping sensitive or copyrighted content.
- Use polite scraping methods (e.g., rate limiting).

- **Tools to Avoid Detection:**

- User-agent rotation.
- Proxy servers.



Why Python for Web Scraping?

- **Ease of Use:**
 - Simple syntax.
- **Libraries:**
 - Requests: For sending HTTP requests.
 - BeautifulSoup: For parsing HTML.
 - Selenium: For handling JavaScript-heavy websites.
- **Community Support:**
 - Extensive documentation and active forums.

Introduction to Flask

- **What is Flask?**
 - A lightweight web framework in Python.
- **Why Use Flask in Web Scraping?**
 - Build APIs to serve scraped data.
 - Create dashboards for data visualization.
 - Automate scraping tasks via web forms.

Workflow of Web Scrapping



1. Identify the Target Website:

1. URL and specific data requirements.

2. Send HTTP Requests:

1. Use requests.get() to fetch the webpage.

3. Parse HTML Content:

1. Extract elements with BeautifulSoup.

4. Store Data:

1. Save in a database or export as CSV/JSON.

5. Build API with Flask:

1. Serve scraped data dynamically.

Hands-On Setup



- **Install Required Libraries:**
 - pip install flask beautifulsoup4 requests
- **Basic Project Structure:**
 - app.py (Flask app)
 - scraper.py (Scraping logic)
 - templates/ (HTML files)

Code for Web Scrapping Logic

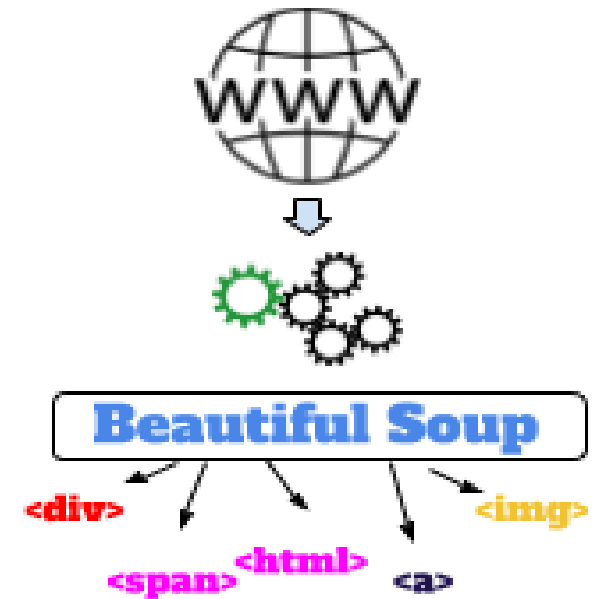


Example: Scrapping Titles from a website

```
import requests

from bs4 import BeautifulSoup

def scrape_blog():
    url = https://example-blog.com
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    titles = [title.text for title in soup.find_all('h2')]
    return titles
```





Web Scraping Code With Flask Integration

- Example:

```
from flask import Flask, jsonify
from scraper import scrape_blog
app = Flask(__name__)
@app.route('/')
def home():
    data = scrape_blog()
    return jsonify(data)
if __name__ == '__main__':
    app.run(debug=True)
```



Demo Time



- **Steps:**

- Run the Flask app: `python app.py`
- Open the browser:
`http://127.0.0.1:5000/`
- View the scraped data in **JSON** format.

Visualization Ideas:

- Display the data in a table or graph.

Challenges in Web Scrapping



- **Dynamic Content:**
 - JavaScript-heavy websites.
- **Anti-Scrapping Mechanisms:**
 - CAPTCHAs, IP blocking.
- **Changing Website Structure:**
 - Frequent updates in HTML layout.



Best Practices

- Use proper headers (e.g., User-Agent).
- Respect robots.txt.
- Cache frequently scraped data.
- Handle errors and timeouts.



User-Agent



Most Common HTTP Headers for Web Scraping

Here's a rundown of the most frequently used HTTP headers for web scraping:

User-Agent

The User-Agent header identifies the browser or tool making the request. It's one of the most important headers because most websites block non-browser user agents. Mimicking a real browser through this header can make your scraper look like legitimate traffic.

Example:

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
Safari/537.36

Example of Robots.txt



← → ↻ nielit.gov.in/chandigarh/robots.txt

```
#
# robots.txt
#
# This file is to prevent the crawling and indexing of certain parts ✓
# of your site by web crawlers and spiders run by sites like Yahoo!
# and Google. By telling these "robots" where not to go on your site, ✓
# you save bandwidth and server resources. ✓
#
# This file will be ignored unless it is at the root of your host:
# Used:      http://example.com/robots.txt
# Ignored:  http://example.com/site/robots.txt
#
# For more information about the robots.txt standard, see:
# http://www.robotstxt.org/robotstxt.html
```



Why Use Cache For Web Scrapping?



Caching increases performance and decreases computing costs by preventing redundancy. Here are a few reasons why you should use cache in web scrapping:

- **Reduce Response Time**

Sending HTTP requests while scraping can be time-consuming, especially with rich and complex web pages. On the other hand, cached data are saved in memory storage and can be retrieved in fractions of a second. This boost in response time speeds up the web scraping process and **accelerates the development and debugging** cycles.

- **Reduce Server Load**

Instead of making repeated requests to the server for the same data, it can be retrieved from the cache. This reduces the number of requests sent, which prevents overloading the websites' servers and results in more ethical web scraping practices.

- **Reduce Consumed Bandwidth**

Using cached data can help minimize bandwidth usage by eliminating the number of repeated requests. This is particularly big when using residential proxies which charge by bandwidth and can be very expensive.

Use retry and timeout strategies

Another way to manage errors when web scraping with Python is to use retry and timeout strategies. Retry and timeout strategies are methods to handle network errors or delays, such as connection errors, server errors, or slow responses, when requesting a web page. Retry strategies allow you to repeat your web scraping requests a certain number of times or until a certain condition is met, in case of failures or errors. Timeout strategies allow you to specify a maximum time or limit for your web scraping requests, in case of delays or hangs. By using retry and timeout strategies, you can increase the success and efficiency of your web scraping requests, as well as avoid wasting resources or time.

Is Web Scraping Legal or Illegal?



- Web scraping legality depends on:

Illegal Scenarios:

- Violating terms of service.
- Bypassing security measures.
- Infringing copyright or privacy laws.
- Causing harm to website servers.

Legal Scenarios:

- Scraping publicly available data.
- Complying with terms of service.
- For research or personal use.
- Using open APIs.

Best Practices:

- Check terms of service.
- Seek permission if unsure.
- Respect robots.txt and server limits.
- Avoid sensitive data scraping.

- **Let's Discuss:**
- Any questions about setup or implementation?
- Real-world use cases for your projects.



Conclusion



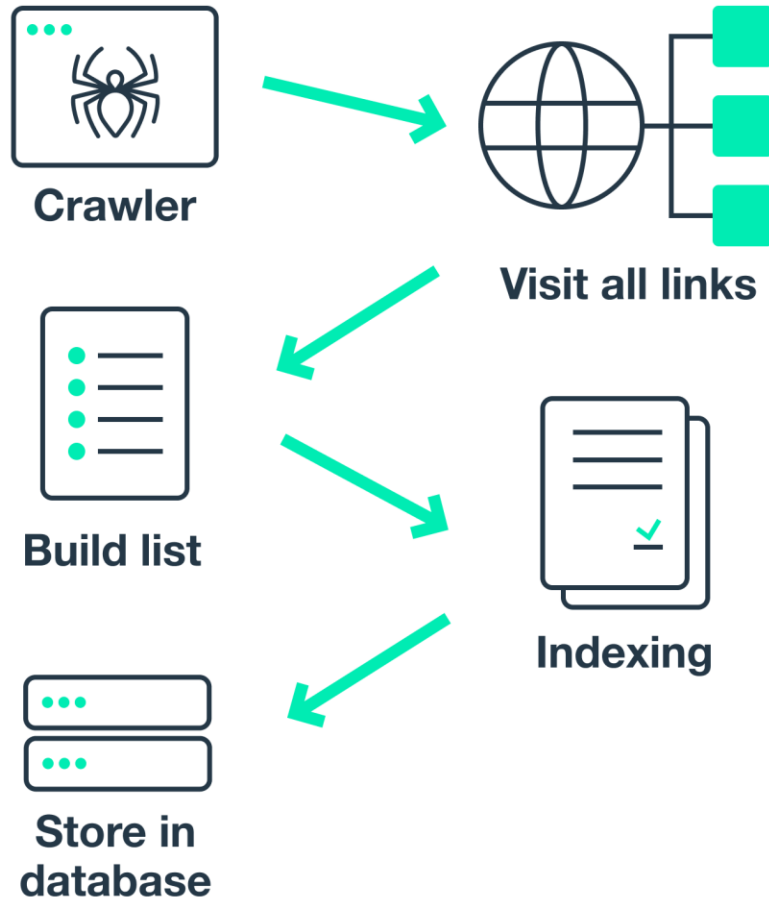
- **Key Takeaways:**

- Python is a powerful tool for web scraping.
- Flask makes it easy to build APIs and serve data.
- Ethical scraping is critical for sustainable use.

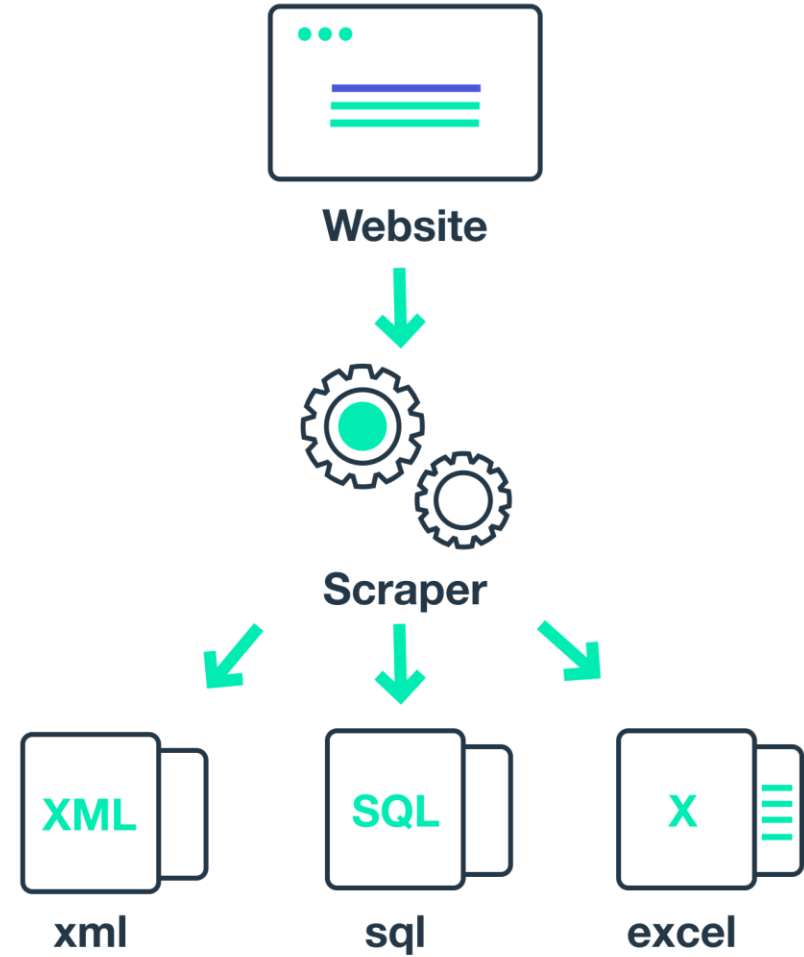
- **Next Steps:**

- Explore advanced techniques for web scraping like: Selenium, Scrapy.
- Implement a complete data pipeline.

Web Crawler



Web Scrapping



Title: Government of India : National Institute of Electronics & Information Technology

Content from: <https://nielit.gov.in/chandigarh/tender>

Scraped Elements for Tag: <p>

- राष्ट्रीय इलेक्ट्रॉनिकी एवं सूचना प्रौद्योगिकी संस्थान ,चंडीगढ़
- National Institute of Electronics & Information Technology, Chandigarh
-
- The last date for submission of technical & financial bid of Tender No: NIELIT/CH/PUR/232/V-4/2024/01 for Hiring of vendor for Videography and Photography services during Examination has been extended upto 16-12-2024 5:00 P.M.
- Limited Tender Document for Videography and Photography services for Examination
- EOI for “Employability Enhancement & Livelihood Training Program [EELTP] of SC/ST & EWS (Women) Youth through Capacity Building and Skill Development in IECT”
-
- EOI for “Employability Enhancement & Livelihood Training Program [EELTP] of SC/ST & EWS (Women) Youth through Capacity Building and Skill Development in IECT”
-
- Tender for Conduct of Physical Efficiency Test & Physical Measurement Test
- EOI for Training Partners for the Aspirational District Project at Moga and Ferozpur
-
- GeM Bid for Hiring of Security Vehicles for Transport of Confidential Material
- Tender for Books
- GeM-Bid for Frisking during Examinations
- GeM bid for Solar Street Lights System

LIVE DEMO:

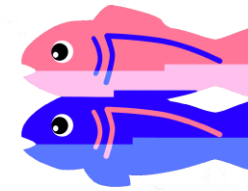
<https://webscrapingexample.glitch.me/>



Flask



BeautifulSoup



Glitch

Create Live Project

NIELIT Chandigarh/Ropar



Web Scraping is the process of extracting and parsing data from websites in an automated fashion using a computer program



Create Webscraping website using Python BeautifulSoup, Flask Framework On [Glitch.com](https://glitch.com)

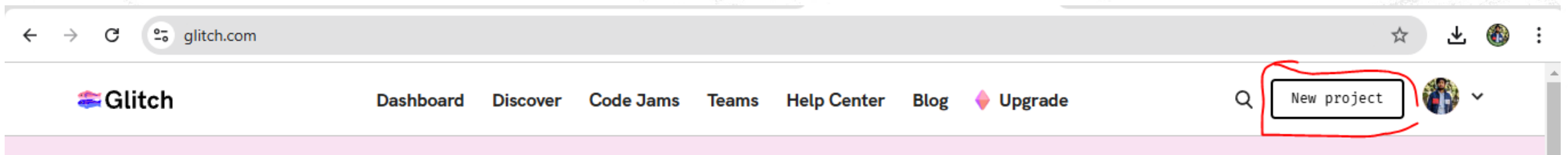


- **Set Up Glitch Project:**

- Sign up or Login on [Glitch.com](https://glitch.com)



- Go to [Glitch.com](https://glitch.com) homepage after logging in and click on **create a new project**.

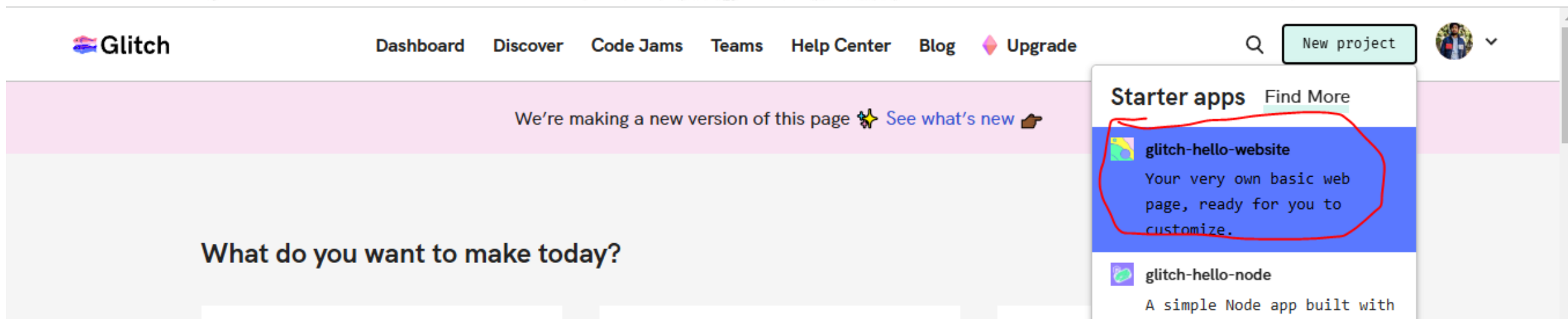




Create Webscraping website using Python BeautifulSoup, Flask Framework On [Glitch.com](https://glitch.com)



- Choose the "Hello World Website" template.

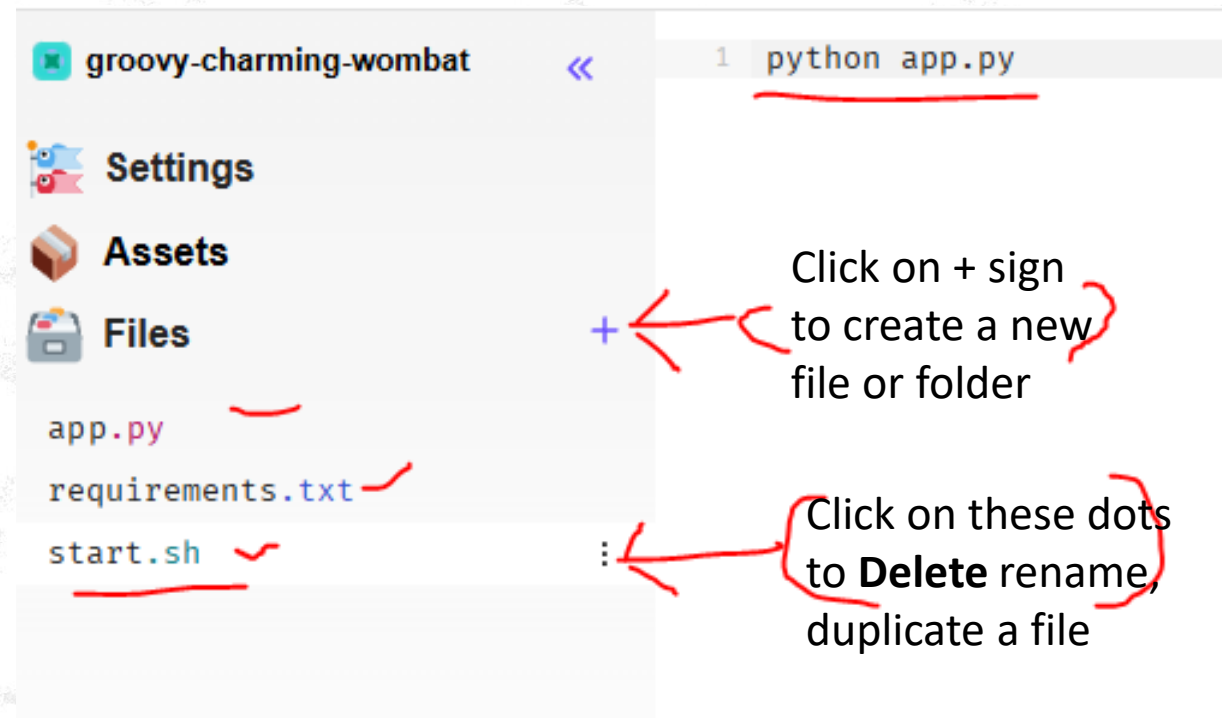


Create Webscraping website using Python BeautifulSoup, Flask Framework On [Glitch.com](https://glitch.com)

Delete all the existing files and Setup up website with Python – Flask by creating all these 3 files listed in below screenshot and in `start.sh` file write `python app.py`

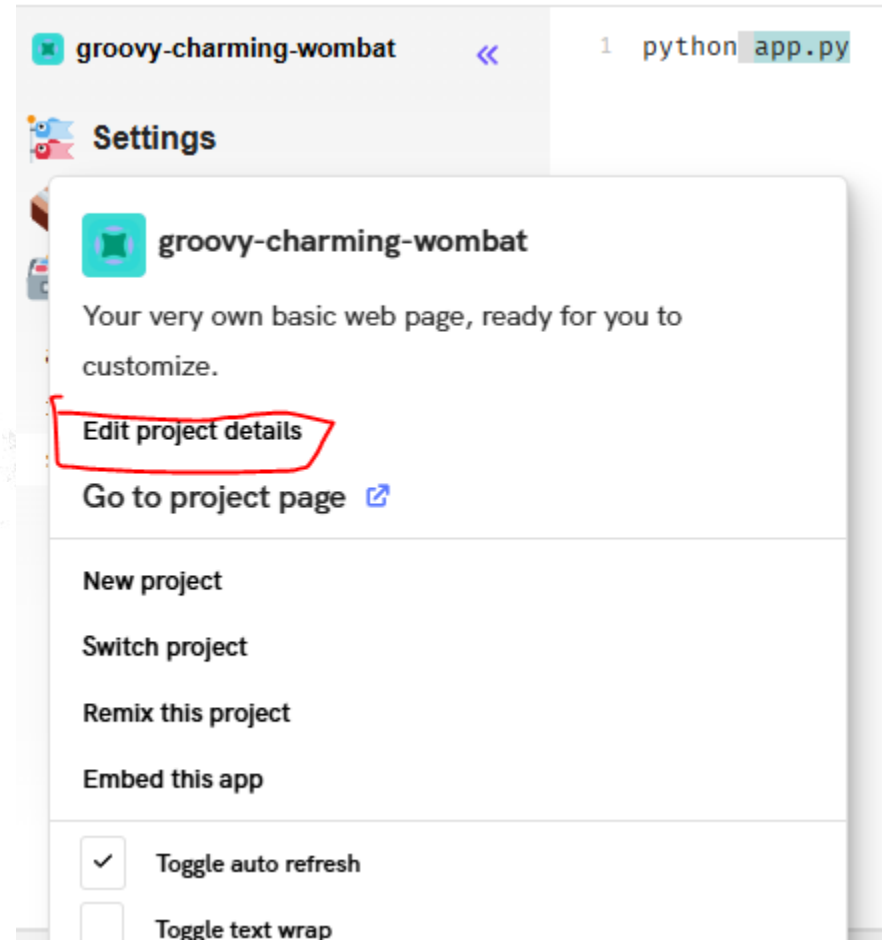
Create These Files:

1. `app.py`
2. `requirements.txt`
3. `start.sh`



Create Webscraping website using Python BeautifulSoup, Flask Framework On [Glitch.com](https://glitch.com)

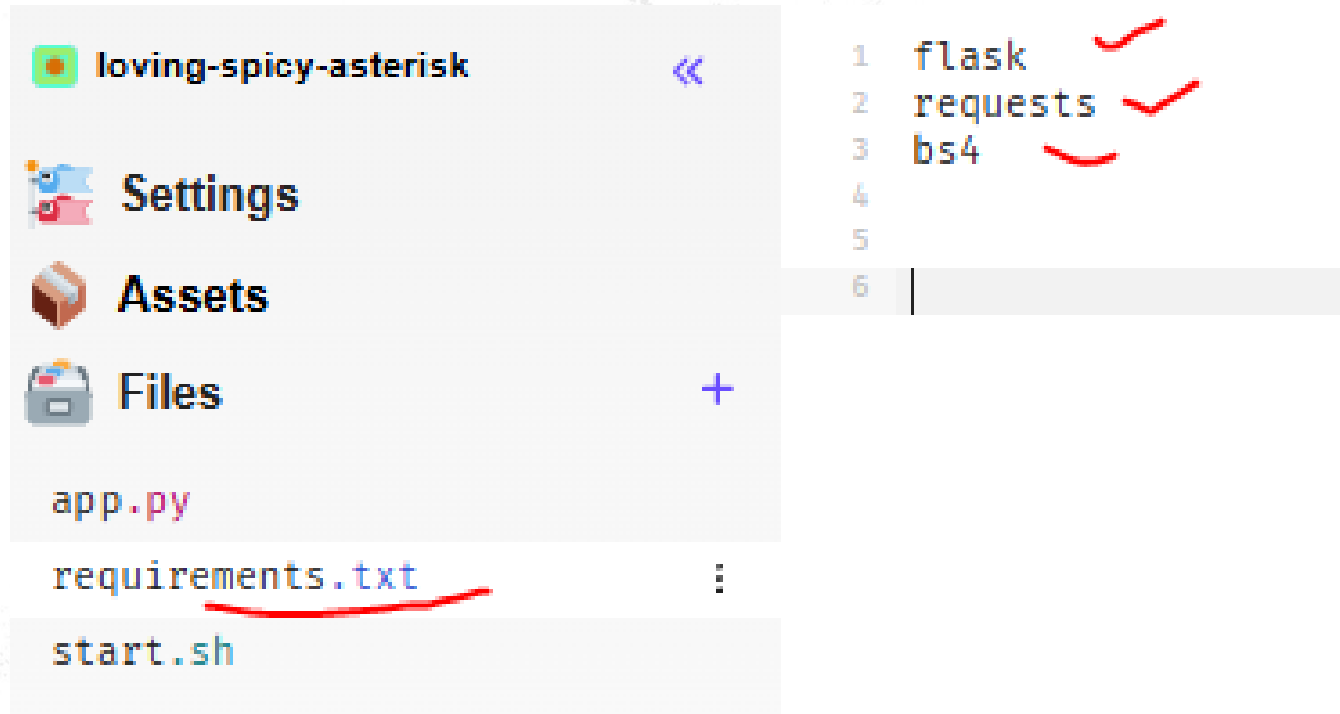
- Click on Settings and then Edit Project Details to rename the project.



Create Webscraping website using Python BeautifulSoup, Flask Framework On [Glitch.com](https://glitch.com)



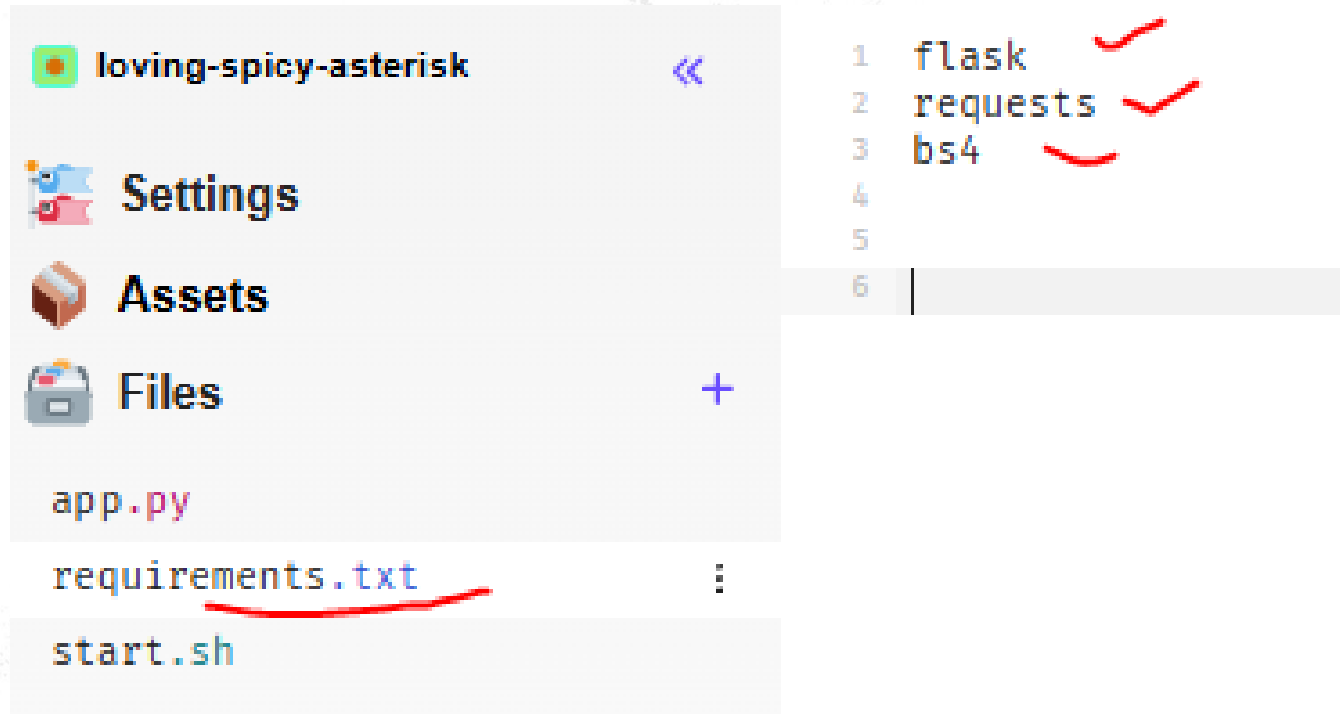
- Add list of all required libraries in **requirements.txt** file.



Create Webscraping website using Python BeautifulSoup, Flask Framework On [Glitch.com](https://glitch.com)



- Add list of all required libraries in **requirements.txt** file.



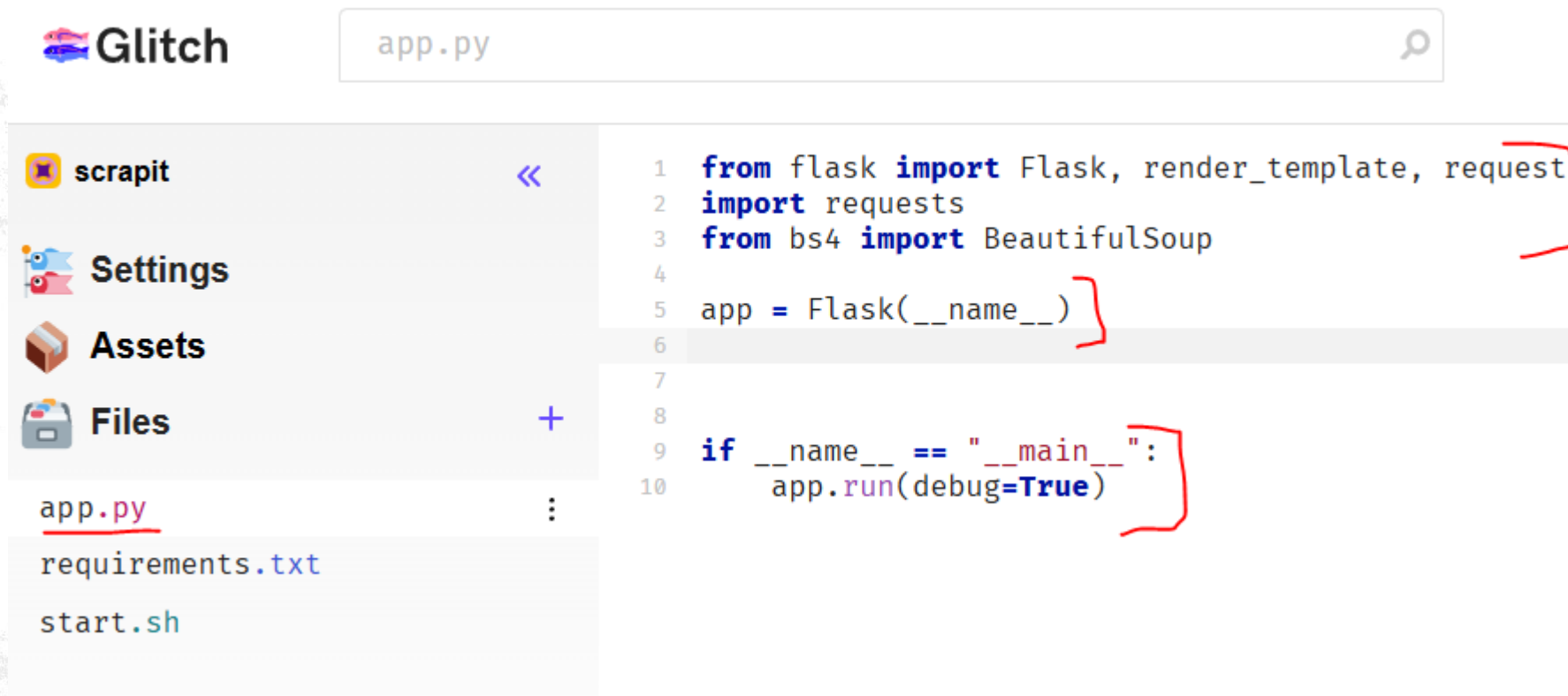
The screenshot shows the Glitch project interface. On the left, the file explorer lists the following files and folders: **loving-spicy-asterisk** (with a double left arrow icon), **Settings** (with a gear icon), **Assets** (with a folder icon), **Files** (with a folder icon and a plus sign), **app.py**, **requirements.txt** (underlined in red), and **start.sh**. On the right, the content of the **requirements.txt** file is displayed in a code editor with line numbers 1 through 6. The code lists the following libraries: **flask** (line 1, with a red checkmark), **requests** (line 2, with a red checkmark), and **bs4** (line 3, with a red checkmark). Lines 4, 5, and 6 are empty.

```
1 flask ✓
2 requests ✓
3 bs4 ✓
4
5
6
```

Create Webscraping website using Python BeautifulSoup, Flask Framework On Glitch.com



- To Setup Your Flask App edit **app.py** file import all the required libraries we're going to use and then set up flask app structure:



The screenshot shows the Glitch.com editor interface. On the left, there's a sidebar with icons for 'scrapit', 'Settings', 'Assets', and 'Files'. Below these, a list of files is shown: 'app.py' (highlighted with a red underline), 'requirements.txt', and 'start.sh'. The main area displays the code in 'app.py'. The code is as follows:

```
1 from flask import Flask, render_template, request
2 import requests
3 from bs4 import BeautifulSoup
4
5 app = Flask(__name__)
6
7
8
9 if __name__ == "__main__":
10     app.run(debug=True)
```

Red brackets are drawn on the code to group imports (lines 1-3) and the main execution block (lines 9-10).

Create Webscraping website using Python BeautifulSoup, Flask Framework On Glitch.com



- Edit **app.py** app all the routes and scraping logic:

```

app = Flask(__name__)

# Home Route - Display the Form
@app.route("/", methods=["GET"])
def index():
    return render_template("index.html")

# Scraping Route - Process URL and Display Results Based on User Input
@app.route("/scrape", methods=["POST"])
def scrape():
    if request.method == "POST":
        # Safely get URL and tag from the Form using .get()
        url = request.form.get("url")
        tag = request.form.get("tag")

        # Check if both URL and tag are provided
        if not url or not tag:
            error_message = "Both URL and Tag are required fields."
            return render_template("result.html", error=error_message)

        try:
            # Custom headers to mimic a real browser request to bypass 403 Client Error: Forbidden for url: such errors
            headers = {
                "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"
            }

            # Fetch page content with custom headers
            response = requests.get(url, headers=headers)
            response.raise_for_status() # Raise exception for invalid responses

            # Parse content with BeautifulSoup
            soup = BeautifulSoup(response.content, "html.parser")

            # Extract content based on user-defined tag
            elements = [element.get_text() for element in soup.find_all(tag)]
            title = soup.title.string if soup.title else "No Title Found"

            return render_template("result.html", title=title, elements=elements, tag=tag, url=url)

        except Exception as e:
            error_message = "An error occurred: {}".format(str(e)) # Using str.format()
            return render_template("result.html", error=error_message)

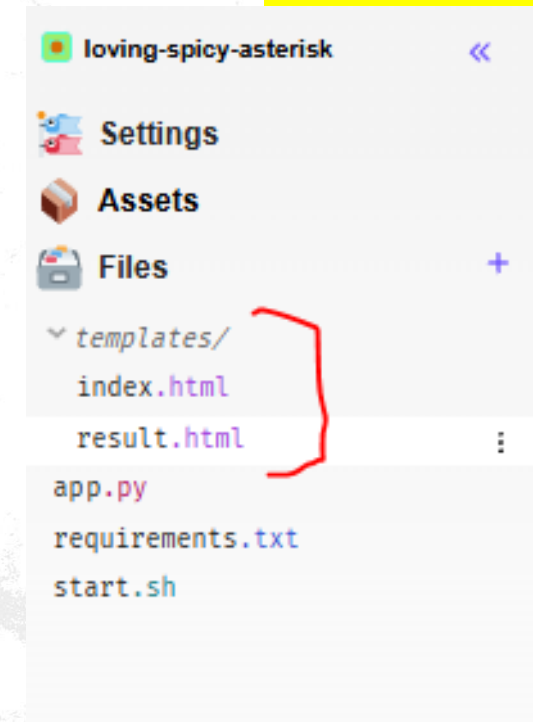
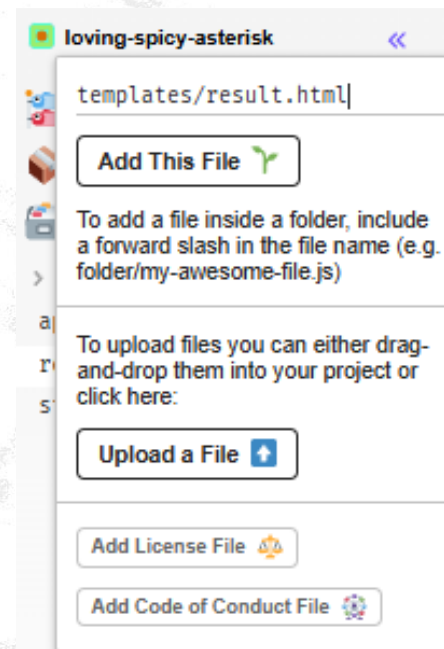
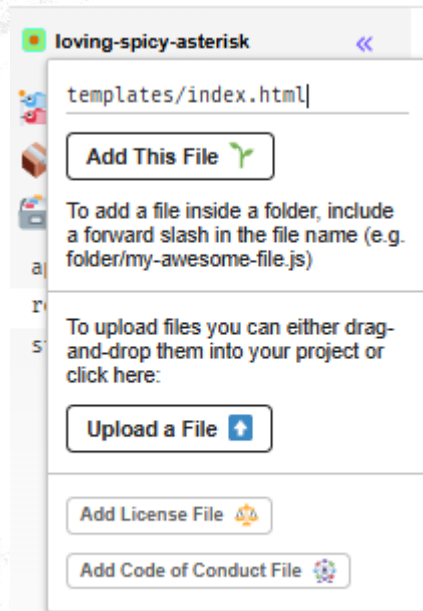
if __name__ == "__main__":
    app.run(debug=True)
  
```



Create Webscraping website using Python BeautifulSoup, Flask Framework On Glitch.com



- Create templates folder and inside it create index.html file and result.html file to do so click on + sign and then enter **templates/index.html** and **templates/result.html** it will create templates folder and inside it **index.html** and **result.html** file





Create Webscraping website using Python BeautifulSoup, Flask Framework On Glitch.com



- Add Following Code in **index.html** file

```
scrapit  <<  index.html  PRETTIER

Settings
Assets
Files +
  templates/
    index.html :
    result.html
  app.py
  requirements.txt
  start.sh

1~ <html>
2~ <head>
3~   <link rel="icon" type="image/png"
4~     href="https://cdn.glitch.global/011875c1-2e8a-4ff4-806a-793934a0acda/android-chrome-512x512.png?v=1734461641548" />
5~   <title>Web Scraper</title>
6~   <style>
7~     body {
8~       font-family: Arial, sans-serif;
9~       margin: 0;
10~      padding: 0;
11~      background-color: #f9f9f9;
12~      text-align: center;
13~    }
14~    .logo {
15~      margin-top: 30px;
16~      width: 128px;
17~      height: 128px;
18~    }
19~  </style>
20~ </head>
21~ <body>
22~   
24~   <h1>Customizable Web Scraper</h1>
25~   <form action="/scrape" method="POST">
26~     <label for="url">Enter URL:</label><br>
27~     <input type="text" id="url" name="url" placeholder="https://example.com" required><br><br>
28~
29~     <label for="tag">Enter Tag to Scrape (e.g., p, h1, img):</label><br>
30~     <input type="text" id="tag" name="tag" placeholder="p" required><br><br>
31~
32~     <button type="submit">Scrape</button>
33~   </form>
34~
35~ </body>
36~ </html>
```

Create Webscraping website using Python BeautifulSoup, Flask Framework On Glitch.com



- Add Following Code in **result.html** file

```

scrapit
<<
result.html
★ PRETTIER

1~ <html>
2~ <head>
3~   <link rel="icon" type="image/png"
4~     href="https://cdn.glitch.global/011875c1-2e8a-4ff4-806a-793934a0acda/android-chrome-512x512.png?v=1734461641548" />
5~   <title>Scraped Results</title>
6~ </head>
7~ <body>
8~   <h1>Scraped Results</h1>
9~   {% if error %}
10~     <p style="color: red;">{{ error }}</p>
11~   {% else %}
12~     <h2>Title: {{ title }}</h2>
13~     <h3>Content from: <a href="{{ url }}" target="_blank">{{ url }}</a></h3>
14~     <h3>Scraped Elements for Tag: &lt;{{ tag }}&gt;</h3>
15~     <ul>
16~       {% for element in elements %}
17~         <li>{{ element }}</li>
18~       {% else %}
19~         <li>No content found for tag &lt;{{ tag }}&gt;.</li>
20~       {% endfor %}
21~     </ul>
22~   {% endif %}
23~   <br>
24~   <a href="/">Go Back</a>
25~ </body>
26~ </html>
27~
  
```

Create Webscraping website using Python BeautifulSoup, Flask Framework On Glitch.com



- Let's Test Our Web Scraping App open your app URL
- Enter URL you want to scrap and enter Tag to Scrap
- Click on Scrape



The screenshot shows a web browser window with the address bar displaying scrapit.glitch.me. The page features the NIELIT logo at the top center. Below the logo, the title "Customizable Web Scraper" is displayed. The interface includes two input fields: "Enter URL:" with the value https://www.punjabkesari.in/, and "Enter Tag to Scrape (e.g., p, h1, img):" with the value p. A red circle highlights the "Scrape" button located below the tag input field.

Enjoy your own Web Scraping App



← → ↻ 📄 scrapit.glitch.me/scrape ☆ 🌐 ⋮

Scraped Results

Title: Latest News, Breaking News Today - Entertainment, Cricket, Business, Politics - India Today

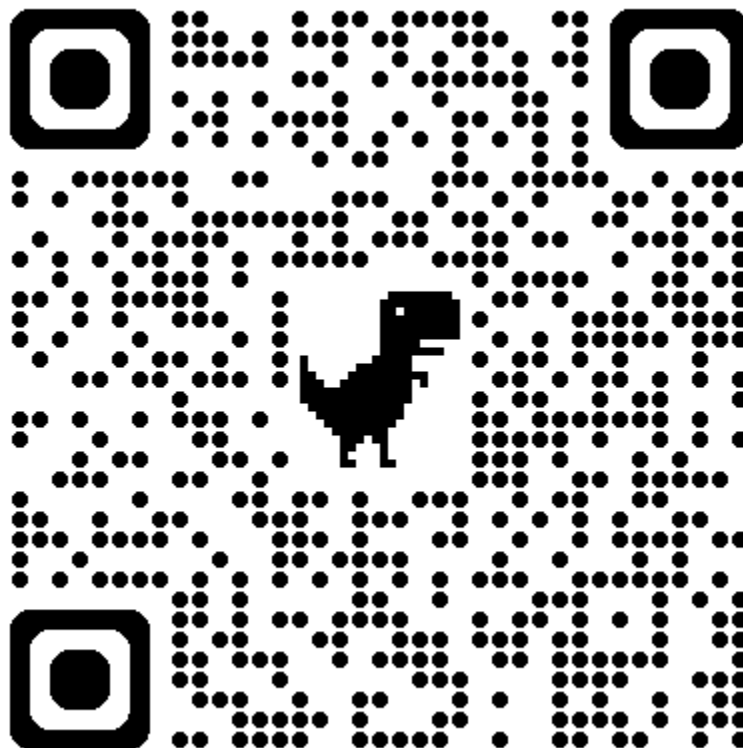
Content from: <https://www.indiatoday.in/>

Scraped Elements for Tag: <p>

- At least 126 people have died after multiple earthquakes struck Tibet. The tremors were also felt in neighbouring Nepal. Rescue efforts are underway as the earthquake reduced several houses to rubble.
- Distinguished Scientist at Isro, Dr V Narayanan is currently the Director of the Liquid Propulsion Systems Centre (LPSC).
- Meta CEO Mark Zuckerberg announced the overhauling of Facebook's censorship policies, removing fact-checkers, and adopting a community-driven moderation system inspired by Elon Musk's X.
- AAP accused the BJP of trying to throw Delhi Chief Minister Atishi out of her official residence. The BJP, while denying the claims, accused Atishi of never moving into the house to avoid upsetting Arvind Kejriwal.
- Ajith Kumar's car crashed during a practice session for the Dubai Grand Prix, but the actor emerged unhurt. Kumar returned to car racing competitions after 15 years, following multiple injuries.
- Pranab Mukherjee's daughter, Sharmistha Mukherjee, expressed her gratitude to Prime Minister Narendra Modi for the decision, calling it a "gracious" and "unexpected" gesture.
- 185 employees have been fired by Apple for misusing a grants clause to gain higher salary from the company. Of these, many are said to be Indians.
- Navy divers from Visakhapatnam were deployed to speed up rescue operations of the labourers trapped in a rat-hole mine in Assam. The teams of the National Disaster Response Force and the State Disaster Response Force are already on the spot.
- OnePlus has launched its latest flagship, the OnePlus 13, in India with a starting price of Rs 69,999. The phone, which will go on sale via Amazon, brings several upgrades.
- Anita Anand, the Indian-origin Transport Minister, is a top contender to succeed Justin Trudeau. Born in Nova Scotia to Indian physician parents, she has held key roles in the incumbent Canadian government.
- Delhi Election Schedule: The Delhi elections will witness a fierce triangular contest between the AAP, BJP and the Congress. The ruling AAP, battling a wave of corruption allegations, will be eyeing a hat-trick.



Live Demo



<https://scrapit.glitch.me/>