**1. Importing libraries**

The program begins by importing three key Python libraries:

* **requests** — used to send HTTP requests and receive responses from web servers.
* **BeautifulSoup** (from the bs4 package) — used to parse and navigate the HTML structure of a webpage.
* **pandas** — used to organize and analyze the extracted data in a tabular format.

**2. Fetching the webpage content**

The program defines the URL of the website (the GitHub Topics page) and uses the requests.get() function to send an HTTP GET request to that URL.  
The server responds with the page’s HTML content.  
The program stores this response in a variable for further use.

**3. Verifying the HTTP request**

Every response from a web server includes a **status code** that indicates whether the request succeeded.  
The program prints this code — it should be 200, meaning *OK*.  
If it were, for example, 404, that would mean *page not found*, and the script would need to handle it differently.  
This check ensures the web request didn’t fail silently.

**4. Previewing the content**

Before proceeding, the script prints the **first 100 characters** of the HTML content to verify that something meaningful (like a valid HTML structure) was received.  
This helps confirm that the request didn’t return an error page or empty data.

**5. Saving the webpage to a file**

The entire HTML content is then written to a local file called **webpage.html**.  
The script uses the same encoding that the server specified in its response, which ensures that characters like symbols or non-English letters appear correctly when opened later.  
This step allows the script (or a human) to inspect the saved page if needed.

**6. Reading and parsing the HTML**

After saving, the script reopens the webpage.html file and loads its content into **BeautifulSoup**.  
BeautifulSoup parses the HTML, creating a structured tree representation of the page — this makes it easy to search for specific tags, classes, and text.

**7. Identifying target data**

The script identifies two types of information to extract from the page:

* **Topic titles** (the names of the topics, such as “Machine Learning” or “Python”).
* **Topic descriptions** (short explanations that appear below each title).

By examining the page structure in a browser’s developer tools, the script determines that titles are contained in <p> tags with a specific class (f3 lh-condensed mb-0 mt-1 Link--primary), and descriptions are in another <p> tag class (f5 color-fg-muted mb-0 mt-1).  
These HTML classes act like unique labels to locate the correct text elements.

**8. Extracting and cleaning the data**

Using BeautifulSoup’s search functions, the script collects all matching elements for titles and for descriptions.  
It then extracts only the **text content** from those elements, removing surrounding whitespace.  
The result is two Python lists — one containing all the topic names, and the other containing all the corresponding descriptions.

**9. Verifying extracted data**

To confirm that the extraction worked correctly, the script prints:

* The number of items in each list (to ensure it matches expectations, like 30 topics).
* The actual contents of both lists (to visually inspect that the right information was captured).

**10. Structuring the data**

Next, the script creates a **dictionary** where each key represents a category of information:  
one for the titles and another for the descriptions.  
The values under these keys are the lists extracted earlier.  
This dictionary format is ideal for converting the data into a structured table later.

**11. Creating a DataFrame**

The program then converts this dictionary into a **pandas DataFrame**, which functions like a spreadsheet in memory.  
Each row represents one GitHub topic, with columns for “Title” and “Description.”  
This makes the data much easier to view, analyze, and export.

**12. Displaying the result**

Finally, the script prints the DataFrame to the console so the user can see a neatly formatted table of the extracted topics and their descriptions.  
This serves as confirmation that the data pipeline — from fetching the web page to parsing and organizing the information — worked as intended.

**Summary of the Workflow**

1. **Connect to the website** → Fetch HTML.
2. **Check the connection** → Ensure response status is 200.
3. **Save locally** → Keep a copy of the page.
4. **Parse the page** → Use BeautifulSoup to interpret HTML.
5. **Locate key data** → Identify HTML tags/classes for topics and descriptions.
6. **Extract and clean** → Pull out the visible text.
7. **Organize results** → Store in a dictionary and convert to a DataFrame.
8. **Display output** → View structured information in a readable table.