

Cheat Sheet: API's and Data Collection

Package/Method	Description	Code Example
Accessing element attribute	Access the value of a specific attribute of an HTML element.	Syntax: <pre>attribute = element[(attribute)]</pre> Example: <pre>href = link_element[(href)]</pre>
BeautifulSoup()	Parse the HTML content of a web page using BeautifulSoup. The parser type can vary based on the project.	Syntax: <pre>soup = BeautifulSoup(html, (html.parser))</pre> Example: <pre>html = (https://api.example.com/data) soup = BeautifulSoup(html, (html.parser))</pre>
delete()	Send a DELETE request to remove data or a resource from the server. DELETE requests delete a specified resource on the server.	Syntax: <pre>response = requests.delete(url)</pre> Example: <pre>response = requests.delete((https://api.example.com/delete))</pre>

find()	Find the first HTML element that matches the specified tag and attributes.	<p>Syntax:</p> <pre>element = soup.find(tag, attrs)</pre> <p>Example:</p> <pre>first_link = soup.find((a), {(class): (link)})</pre>
find_all()	Find all HTML elements that match the specified tag and attributes.	<p>Syntax:</p> <pre>elements = soup.find_all(tag, attrs)</pre> <p>Example:</p> <pre>all_links = soup.find_all((a), {(class): (link)})</td></pre>
findChildren()	Find all child elements of an HTML element.	<p>Syntax:</p> <pre>children = element.findChildren()</pre> <p>Example:</p> <pre>child_elements = parent_div.findChildren()</pre>
get()	Perform a GET request to retrieve data from a specified URL.	<p>Syntax:</p> <pre>response = requests.get(url)</pre>

	<p>GET requests are typically used for reading data from an API. The response variable will contain the server's response, which you can process further.</p>	<p>Example:</p> <pre>response = requests.get((https://api.example.com/data))</pre>
Headers	<p>Include custom headers in the request. Headers can provide additional information to the server, such as authentication tokens or content types.</p>	<p>Syntax:</p> <pre>headers = {(HeaderName): (Value)}</pre> <p>Example:</p> <pre>base_url = (https://api.example.com/data) headers = {(Authorization): (Bearer YOUR_TOKEN)} response = re</pre>
Import Libraries	<p>Import the necessary Python libraries for web scraping.</p>	<p>Syntax:</p> <pre>from bs4 import BeautifulSoup</pre>
json()	<p>Parse JSON data from the response. This extracts and works with the data returned by the API. The response.json() method converts the JSON response into a Python data structure (usually a dictionary or list).</p>	<p>Syntax:</p> <pre>data = response.json()</pre> <p>Example:</p> <pre>response = requests.get((https://api.example.com/data)) data = response.json()</pre>

next_sibling()	Find the next sibling element in the DOM.	<p>Syntax:</p> <pre>sibling = element.find_next_sibling()</pre> <p>Example:</p> <pre>next_sibling = current_element.find_next_sibling()</pre>
parent	Access the parent element in the Document Object Model (DOM).	<p>Syntax:</p> <pre>parent = element.parent</pre> <p>Example:</p> <pre>parent_div = paragraph.parent</pre>
post()	Send a POST request to a specified URL with data. Create or update POST requests using resources on the server. The data parameter contains the data to send to the server, often in JSON format.	<p>Syntax:</p> <pre>response = requests.post(url, data)</pre> <p>Example:</p> <pre>response = requests.post((https://api.example.com/submit), data={{key): (value}})</pre>
put()	Send a PUT request to update data on	<p>Syntax:</p> <pre>response = requests.put(url, data)</pre>

	<p>the server. PUT requests are used to update an existing resource on the server with the data provided in the data parameter, typically in JSON format.</p>	<p>Example:</p> <pre>response = requests.put((https://api.example.com/update), data={{key): (value}})</pre>
Query parameters	<p>Pass query parameters in the URL to filter or customize the request. Query parameters specify conditions or limits for the requested data.</p>	<p>Syntax:</p> <pre>params = {(param_name): (value)}</pre> <p>Example:</p> <pre>base_url = "https://api.example.com/data" params = {"page": 1, "per_page": 10} response = requests.get(base_url, params=params)</pre>
select()	<p>Select HTML elements from the parsed HTML using a CSS selector.</p>	<p>Syntax:</p> <pre>element = soup.select(selector)</pre> <p>Example:</p> <pre>titles = soup.select((h1))</pre>
status_code	<p>Check the HTTP status code of the response. The HTTP status code indicates the result of</p>	<p>Syntax:</p> <pre>response.status_code</pre>

	<p>the request (success, error, redirection). Use the HTTP status codeIt can be used for error handling and decision-making in your code.</p>	<p>Example:</p> <pre>url = "https://api.example.com/data" response = requests.get(url) status_code = response.status_code</pre>
tags for find() and find_all()	<p>Specify any valid HTML tag as the tag parameter to search for elements of that type. Here are some common HTML tags that you can use with the tag parameter.</p>	<p>Tag Example:</p> <ul style="list-style-type: none">- (a): Find anchor () tags.- (p): Find paragraph ((p)) tags.- (h1), (h2), (h3), (h4), (h5), (h6): Find heading tags from level 1 to 6 ((h1),n (h2)).- (table): Find table () tags.- (tr): Find table row () tags.- (td): Find table cell ((td)) tags.- (th): Find table header cell ((td))tags.- (img): Find image ((img)) tags.- (form): Find form ((form)) tags.- (button): Find button ((button)) tags.
text	<p>Retrieve the text content of an HTML element.</p>	<p>Syntax:</p> <pre>text = element.text</pre> <p>Example:</p> <pre>title_text = title_element.text</pre>

