Web scraping in python

**How to Make API Call Using Python**

APIs ([Application Programming Interfaces](https://www.geeksforgeeks.org/what-is-an-api/)) are an essential part of modern software development, allowing different applications to communicate and share data. Python provides a popular library i.e. requests library that simplifies the process of calling API in Python. In this article, we will see how to make API calls in [Python](https://www.geeksforgeeks.org/python-programming-language-tutorial/).

**Make API Call in Python**

Below, is the step-by-step code explanation and example of how to make a Python API call:

**Step 1: Install the Library**

The [requests](https://www.geeksforgeeks.org/python-requests-tutorial/) library simplifies the process of making HTTP requests, including [GET](https://www.geeksforgeeks.org/get-method-python-requests/), [POST](https://www.geeksforgeeks.org/what-is-postpower-on-self-test/), PUT, DELETE, etc., which are commonly used in API interactions. To install the request library use the below command.

pip install requests

**Step 2: Making a GET request**

Below, the code defines a function**get\_posts()**to fetch posts from a specified API endpoint. It uses the requests library to make a GET request to the API URL. If the request is successful (status code 200), it converts the response to JSON format and returns the list of posts.

**def** get\_posts():

*# Define the API endpoint URL*

url = 'https://jsonplaceholder.typicode.com/posts'

**try**:

*# Make a GET request to the API endpoint using requests.get()*

response = requests.get(url)

*# Check if the request was successful (status code 200)*

**if** response.status\_code == 200:

posts = response.json()

**return** posts

**else**:

print('Error:', response.status\_code)

**return** **None**

**Step 3: Handling Errors**

Below, code adds exception handling for network-related errors during the GET request to the API endpoint. If such an error occurs, it prints an error message and returns None.

**except** requests.exceptions.RequestException **as** e:

*# Handle any network-related errors or exceptions*

print('Error:', e)

**return** **None**

**Step 4: Make API calls**

In Below code , the main() function shows to making an API call by fetching posts from the API using the get\_posts() function. If posts are successfully retrieved, it prints the title and body of the first post. Otherwise, it prints a failure message.

**def** main():

posts = get\_posts()

**if** posts:

print('First Post Title:', posts[0]['title'])

print('First Post Body:', posts[0]['body'])

**else**:

print('Failed to fetch posts from API.')

**if** \_\_name\_\_ == '\_\_main\_\_':

main()

**Complete Code**

Below is the complete code implementation that we have used in **main.py** file to make API call Python.

**import** **requests**

**def** get\_posts():

url = 'https://jsonplaceholder.typicode.com/posts'

**try**:

response = requests.get(url)

**if** response.status\_code == 200:

posts = response.json()

**return** posts

**else**:

print('Error:', response.status\_code)

**return** **None**

**except** requests.exceptions.RequestException **as** e:

print('Error:', e)

**return** **None**

**def** main():

posts = get\_posts()

**if** posts:

print('First Post Title:', posts[0]['title'])

print('First Post Body:', posts[0]['body'])

**else**:

print('Failed to fetch posts from API.')

**if** \_\_name\_\_ == '\_\_main\_\_':

main()

**Output:**

First Post Title: sunt aut facere repellat provident occaecati excepturi optio reprehenderit

First Post Body: quia et suscipit

suscipit recusandae consequuntur expedita et cum

reprehenderit molestiae ut ut quas totam

nostrum rerum est autem sunt rem eveniet architecto

**Python Web Scraping Tutorial**

In today’s digital world, data is the key to unlocking valuable insights, and much of this data is available on the web. But how do you gather large amounts of data from websites efficiently? That’s where **Python web scraping** comes in.Web scraping, the process of extracting data from websites, has emerged as a powerful technique to gather information from the vast expanse of the internet.

In this tutorial, we'll explore various Python libraries and modules commonly used for web scraping and delve into why Python 3 is the preferred choice for this task. Along with this you will also explore how to use powerful tools like **BeautifulSoup**, **Scrapy**, and **Selenium** to scrape any website.

**Essential Packages and Tools for Python Web Scraping**

The latest version of [Python](https://www.geeksforgeeks.org/python-programming-language), offers a rich set of tools and libraries specifically designed for web scraping, making it easier than ever to retrieve data from the web efficiently and effectively.

**Table of Content**

* [Requests Module](https://www.geeksforgeeks.org/python-web-scraping-tutorial/#requests-module)
* [BeautifulSoup Library](https://www.geeksforgeeks.org/python-web-scraping-tutorial/#beautifulsoup-library)
* [Selenium](https://www.geeksforgeeks.org/python-web-scraping-tutorial/#selenium)
* [Lxml](https://www.geeksforgeeks.org/python-web-scraping-tutorial/#lxml)
* [Urllib Module](https://www.geeksforgeeks.org/python-web-scraping-tutorial/#urllib-module)
* [PyautoGUI](https://www.geeksforgeeks.org/python-web-scraping-tutorial/#pyautogui)
* [Schedule](https://www.geeksforgeeks.org/python-web-scraping-tutorial/#schedule)
* [Why Python3 for Web Scraping?](https://www.geeksforgeeks.org/python-web-scraping-tutorial/#why-python3-for-web-scraping)

**Requests Module**

The requests library is used for making HTTP requests to a specific URL and returns the response. Python requests provide inbuilt functionalities for managing both the request and response.

pip install requests

**Example: Making a Request**

Python requests module has several built-in methods to make HTTP requests to specified URI using GET, POST, PUT, PATCH, or HEAD requests. A HTTP request is meant to either retrieve data from a specified URI or to push data to a server. It works as a request-response protocol between a client and a server. Here we will be using the GET request. The [GET method](https://www.geeksforgeeks.org/get-method-python-requests)is used to retrieve information from the given server using a given URI. The GET method sends the encoded user information appended to the page request.

**import** **requests**

*# Making a GET request*

r = requests.get('https://www.geeksforgeeks.org/python-programming-language/')

*# check status code for response received*

*# success code - 200*

print(r)

*# print content of request*

print(r.content)

For more information, refer to our [Python Requests Tutorial](https://www.geeksforgeeks.org/python-requests-tutorial).

**BeautifulSoup Library**

Beautiful Soup provides a few simple methods and Pythonic phrases for guiding, searching, and changing a parse tree: a toolkit for studying a document and removing what you need. It doesn't take much code to document an application.

Beautiful Soup automatically converts incoming records to Unicode and outgoing forms to UTF-8. You don't have to think about encodings unless the document doesn't define an encoding, and Beautiful Soup can't catch one. Then you just have to choose the original encoding. Beautiful Soup sits on top of famous Python parsers like LXML and HTML, allowing you to try different parsing strategies or trade speed for flexibility.

pip install beautifulsoup4

**Example**

1. **Importing Libraries:**The code imports the requests library for making HTTP requests and the BeautifulSoup class from the bs4 library for parsing HTML.
2. **Making a GET Request:**It sends a GET request to 'https://www.geeksforgeeks.org/python-programming-language/' and stores the response in the variable r.
3. **Checking Status Code:**It prints the status code of the response, typically 200 for success.
4. **Parsing the HTML**: The HTML content of the response is parsed using BeautifulSoup and stored in the variable soup.
5. **Printing the Prettified HTML:**It prints the prettified version of the parsed HTML content for readability and analysis.

**import** **requests**

**from** **bs4** **import** BeautifulSoup

*# Making a GET request*

r = requests.get('https://www.geeksforgeeks.org/python-programming-language/')

*# check status code for response received*

*# success code - 200*

print(r)

*# Parsing the HTML*

soup = BeautifulSoup(r.content, 'html.parser')

print(soup.prettify())

**Finding Elements by Class**

Now, we would like to extract some useful data from the HTML content. The soup object contains all the data in the nested structure which could be programmatically extracted.

**import** **requests**

**from** **bs4** **import** BeautifulSoup

*# Making a GET request*

r = requests.get('https://www.geeksforgeeks.org/python-programming-language/')

*# Parsing the HTML*

soup = BeautifulSoup(r.content, 'html.parser')

s = soup.find('div', class\_='entry-content')

content = soup.find\_all('p')

print(content)

**Selenium**

Selenium is a popular Python module used for automating web browsers. It allows developers to control web browsers programmatically, enabling tasks such as web scraping, automated testing, and web application interaction. Selenium supports various web browsers, including Chrome, Firefox, Safari, and Edge, making it a versatile tool for browser automation.

**Example 1: For Firefox**

In this specific example, we're directing the browser to the Google search page with the query parameter "geeksforgeeks". The browser will load this page, and we can then proceed to interact with it programmatically using Selenium. This interaction could involve tasks like extracting search results, clicking on links, or scraping specific content from the page.

*# import webdriver*

**from** **selenium** **import** webdriver

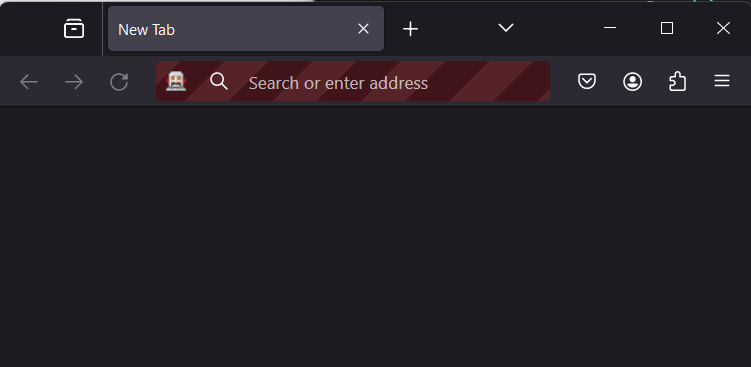
*# create webdriver object*

driver = webdriver.Firefox()

*# get google.co.in*

driver.get("https://google.co.in / search?q = geeksforgeeks")

**Output**



**Example 2: For Chrome**

1. We import the webdriver module from the Selenium library.
2. We specify the path to the web driver executable. You need to download the appropriate driver for your browser and provide the path to it. In this example, we're using the Chrome driver.
3. We create a new instance of the web browser using webdriver.Chrome() and pass the path to the Chrome driver executable as an argument.
4. We navigate to a webpage by calling the get() method on the browser object and passing the URL of the webpage.
5. We extract information from the webpage using various methods provided by Selenium. In this example, we retrieve the page title using the title attribute of the browser object.
6. Finally, we close the browser using the quit() method.

*# importing necessary packages*

**from** **selenium** **import** webdriver

**from** **selenium.webdriver.common.by** **import** By

**from** **webdriver\_manager.chrome** **import** ChromeDriverManager

*# for holding the resultant list*

element\_list = []

**for** page **in** range(1, 3, 1):

page\_url = "https://webscraper.io/test-sites/e-commerce/static/computers/laptops?page=" + str(page)

driver = webdriver.Chrome(ChromeDriverManager().install())

driver.get(page\_url)

title = driver.find\_elements(By.CLASS\_NAME, "title")

price = driver.find\_elements(By.CLASS\_NAME, "price")

description = driver.find\_elements(By.CLASS\_NAME, "description")

rating = driver.find\_elements(By.CLASS\_NAME, "ratings")

**for** i **in** range(len(title)):

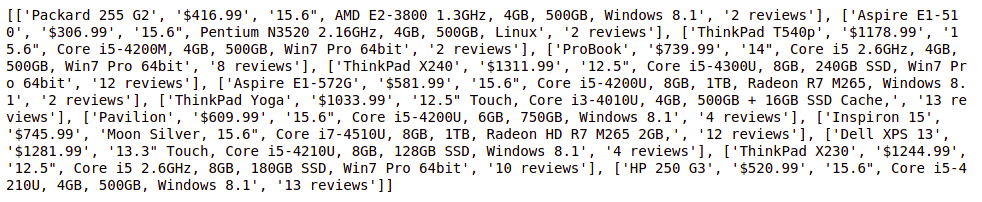
element\_list.append([title[i].text, price[i].text, description[i].text, rating[i].text])

print(element\_list)

*#closing the driver*

driver.close()

**Output**



For more information, refer to our [Python Selenium](https://www.geeksforgeeks.org/selenium-python-tutorial).

**Lxml**

The lxml module in Python is a powerful library for processing XML and HTML documents. It provides a high-performance XML and HTML parsing capabilities along with a simple and Pythonic API. lxml is widely used in Python web scraping due to its speed, flexibility, and ease of use.

pip install lxml

**Example**

Here's a simple example demonstrating how to use the lxml module for Python web scraping:

1. We import the html module from lxml along with the requests module for sending HTTP requests.
2. We define the URL of the website we want to scrape.
3. We send an HTTP GET request to the website using the requests.get() function and retrieve the HTML content of the page.
4. We parse the HTML content using the html.fromstring() function from lxml, which returns an HTML element tree.
5. We use XPath expressions to extract specific elements from the HTML tree. In this case, we're extracting the text content of all the <a> (anchor) elements on the page.
6. We iterate over the extracted link titles and print them out.

from lxml import html

import requests

# Define the URL of the website to scrape

url = 'https://example.com'

# Send an HTTP request to the website and retrieve the HTML content

response = requests.get(url)

# Parse the HTML content using lxml

tree = html.fromstring(response.content)

# Extract specific elements from the HTML tree using XPath

# For example, let's extract the titles of all the links on the page

link\_titles = tree.xpath('//a/text()')

# Print the extracted link titles

for title in link\_titles:

print(title)

**Output**

More information...

**Urllib Module**

The urllib module in Python is a built-in library that provides functions for working with URLs. It allows you to interact with web pages by fetching URLs (Uniform Resource Locators), opening and reading data from them, and performing other URL-related tasks like encoding and parsing. Urllib is a package that collects several modules for working with URLs, such as:

* urllib.request for opening and reading.
* urllib.parse for parsing URLs
* urllib.error for the exceptions raised
* urllib.robotparser for parsing robot.txt files

If urllib is not present in your environment, execute the below code to install it.

pip install urllib3

**Example**

Here's a simple example demonstrating how to use the urllib module to fetch the content of a web page:

1. We define the URL of the web page we want to fetch.
2. We use urllib.request.urlopen() function to open the URL and obtain a response object.
3. We read the content of the response object using the read() method.
4. Since the content is returned as bytes, we decode it to a string using the decode() method with 'utf-8' encoding.
5. Finally, we print the HTML content of the web page.

1

import urllib.request

# URL of the web page to fetch

url = 'https://www.example.com'

try:

# Open the URL and read its content

response = urllib.request.urlopen(url)

# Read the content of the response

data = response.read()

# Decode the data (if it's in bytes) to a string

html\_content = data.decode('utf-8')

# Print the HTML content of the web page

print(html\_content)

except Exception as e:

print("Error fetching URL:", e)

**Output**

**Why Python3 for Web Scraping?**

Python's popularity for web scraping stems from several factors:

**Ease of Use**: Python's clean and readable syntax makes it easy to understand and write code, even for beginners. This simplicity accelerates the development process and reduces the learning curve for web scraping tasks.

**Rich Ecosystem**: Python boasts a vast ecosystem of libraries and frameworks tailored for web scraping. Libraries like BeautifulSoup, Scrapy, and Requests simplify the process of parsing HTML, making data extraction a breeze.

**Versatility**: Python is a versatile language that can be used for a wide range of tasks beyond web scraping. Its flexibility allows developers to integrate web scraping seamlessly into larger projects, such as data analysis, machine learning, or web development.

**Community Support**: Python has a large and active community of developers who contribute to its libraries and provide support through forums, tutorials, and documentation. This wealth of resources ensures that developers have access to assistance and guidance when tackling web scraping challenges.

**Conclusion**

this tutorial has shown you the basics of how to use Python for web scraping. With the tools we’ve discussed, you can start collecting data from the internet quickly and easily. Whether you need this data for a project, research, or just for fun, Python makes it possible. Remember to always scrape data responsibly and follow the rules set by websites. If you're excited to learn more about Python and web scraping, check out our [Python Course](https://gfgcdn.com/tu/Qx7/). It's a great resource to deepen your understanding and enhance your skills, all while having fun exploring the power of Python.

Python Web Scraping - FAQs

**1. What is Python web scraping?**

*Python web scraping refers to the process of extracting data from websites using Python programming. It involves fetching HTML content from a web page and parsing it to gather specific information.*

**2. Is web scraping legal?**

*Web scraping is legal as long as you comply with the website’s* ***terms of service*** *and avoid scraping personal or sensitive data. Always check the site's* ***robots.txt*** *file to ensure you're following the rules.*

**3. What is the difference between BeautifulSoup and Scrapy?**

*BeautifulSoup is a simpler library for beginners focused on HTML parsing and extraction, whereas* ***Scrapy*** *is a more advanced web scraping framework that can handle complex tasks like crawling large datasets or handling pagination automatically.*

**4. What are some common use cases for Python web scraping?**

*Common use cases include extracting data for* ***price comparison****,* ***content aggregation****,* ***job listings****,* ***real estate data****, and* ***sentiment analysis****. Web scraping helps gather structured data from websites for various business and research purposes.*

**Beautifulsoup - Kinds of objects**

**Prerequisites:** [BeautifulSoup](https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/)

In this article, we will discuss different types of objects in Beautifulsoup. When the string or HTML document is given in the constructor of BeautifulSoup, this constructor converts this document to different python objects.

**The four major and important objects are :**

1. BeautifulSoup
2. Tag
3. NavigableString
4. Comments

**1. [BeautifulSoup Object:](https://www.geeksforgeeks.org/beautifulsoup-object-python-beautifulsoup/)**  The BeautifulSoup object represents the parsed document as a whole. So, it is the complete document which we are trying to scrape. For most purposes, you can treat it as a Tag object.

*# importing the module*

**from** **bs4** **import** BeautifulSoup

*# parsing the document*

soup = BeautifulSoup('''<h1>Geeks for Geeks</h1>''',

"html.parser")

print(type(soup))

**Output:**

<class 'bs4.BeautifulSoup'>

**2.**[**Tag Object:**](https://www.geeksforgeeks.org/tag-object-python-beautifulsoup/)Tag object corresponds to an XML or HTML tag in the original document. Further, this object is usually used to extract a tag from the whole HTML document. Further, Beautiful Soup is not an HTTP client which means to scrap online websites you first have to download them using the requests module and then serve them to Beautiful Soup for scraping. Additionally, this object returns the first found tag if your document has multiple tags with the same name.

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Initialize the object with an HTML page*

soup = BeautifulSoup('''

<html>

<b>Geeks for Geeks</b>

</html>

''', "html.parser")

*# Get the tag*

tag = soup.b

*# Print the output*

print(type(tag))

**Output:**

<class 'bs4.element.Tag'>

**We can also change the name of the tag.**

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Initialize the object with an HTML page*

soup = BeautifulSoup('''

<html>

<b>Geeks for Geeks</b>

</html>

''', "html.parser")

*# Get the tag*

tag = soup.b

*# Print the output*

print(tag.name)

*# changing the tag*

tag.name = "Strong"

print(tag)

**Output:**

b

<Strong>Geeks for Geeks</Strong>

**# Attributes :**

**Example 1:**Anything that is NOT tag, is basically an attribute and must contain a value. A tag object can have many attributes and can be accessed either through accessing the keys or directly accessing through value. We can also modify the attributes and their value.

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Initialize the object with an HTML page*

soup = BeautifulSoup('''

<html>

<b class="gfg">Geeks for Geeks</b>

</html>

''', "html.parser")

*# Get the tag*

tag = soup.b

print(tag["class"])

*# modifying class*

tag["class"] = "geeks"

print(tag)

*# delete the class attributes*

**del** tag["class"]

print(tag)

**Output:**

['gfg']

<b class="geeks">Geeks for Geeks</b>

<b>Geeks for Geeks</b>

**Example 2:**A document may contain multi-valued attributes and can be accessed using key-value pair.

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Initialize the object with an HTML page*

*# soup for multi\_valued attributes*

soup = BeautifulSoup('''

<html>

<b class="gfg geeks">Geeks for Geeks</b>

</html>

''', "html.parser")

*# Get the tag*

tag = soup.b

print(tag["class"])

**Output:**

['gfg', 'geeks']

**3. [NavigableString Object:](https://www.geeksforgeeks.org/navigablestring-class-python-beautifulsoup/)**A string corresponds to a bit of text within a tag. Beautiful Soup uses the NavigableString class to contain these bits of text.

***Syntax:****<tag> String here </tag>*

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Initialize the object with an HTML page*

soup = BeautifulSoup('''

<html>

<b>Geeks for Geeks</b>

</html>

''', "html.parser")

*# Get the tag*

tag = soup.b

*# Get the string inside the tag*

string = tag.string

*# Print the output*

print(type(string))

**Output:**

<class 'bs4.element.NavigableString'>

**4.**[**Comment Object:**](https://www.geeksforgeeks.org/how-to-scrape-comment-using-beautifulsoup-in-python/)The Comment object is just a special type of NavigableString and is used to make the codebase more readable.

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Create the document*

markup = "<b><!-- COMMENT --></b>"

*# Initialize the object with the document*

soup = BeautifulSoup(markup, "html.parser")

*# Get the whole comment inside b tag*

comment = soup.b.string

*# Print the type of the comment*

print(type(comment))

**Prerequisite:** [BeautifulSoup](https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/)

**BeautifulSoup(bs4)**is a Python library for pulling data out of HTML and XML files. This module does not come in built-in with Python. To install this type the below command in the terminal. In this article, we will learn about siblings in HTML tags using BeautifulSoup.

Here we will discuss these four sibling properties:

* **previous\_sibling**is used to find the previous element of the given element
* **next\_sibling**is used to find the next element of the given element
* **previous\_siblings**is used to find all previous element of the given element
* **next\_siblings**is used to find all next element of the given element

**Approach**

* Import module
* Load or create HTML code
* Parse HTML code
* Print required sibling.

**Example 1:**To print next immediate sibling

*# Import Module*

**from** **bs4** **import** BeautifulSoup

*# HTML CODE*

html\_code = """<a><b>text1</b><c>text2</c></a>"""

*# Parse HTML CODE*

soup = BeautifulSoup(html\_code, 'html.parser')

*# next element*

print(soup.b.next\_sibling)

**Output:**

*<c>text2</c>*

**Example 2:**To get previous immediate sibling

*# Import Module*

**from** **bs4** **import** BeautifulSoup

*# HTML CODE*

html\_code = """

<a>

<b>text1</b>

<c>text2</c>

</a>"""

*# Parse HTML CODE*

soup = BeautifulSoup(html\_code, 'html.parser')

*# previous element*

print(soup.c.previous\_sibling)

**Output:**

*<b>text1</b>*

Suppose we want to find all the next elements of a tag. For that, we just simply iterate through siblings and print the required tag.

**Example 3:** To get all siblings next to the tag

*# Import Module*

**from** **bs4** **import** BeautifulSoup

*# HTML CODE*

html\_code = """<a><b>text1</b><d>text3</d><c>text2</c></a>"""

*# Parse HTML CODE*

soup = BeautifulSoup(html\_code, 'html.parser')

*# next element*

**for** element **in** soup.b.next\_siblings:

print(element)

**Output:**

*<d>text3</d>*

*<c>text2</c>*

**Example 4:**To get all previous siblings

*# Import Module*

**from** **bs4** **import** BeautifulSoup

*# HTML CODE*

html\_code = """<a><b>text1</b><d>text3</d><c>text2</c></a>"""

*# Parse HTML CODE*

soup = BeautifulSoup(html\_code, 'html.parser')

*# previous element*

**for** element **in** soup.c.previous\_siblings:

print(element)

**Output:**

*<d>text3</d>*

*<b>text1</b>*

This module does not come built-in with Python. To install this type the below command in the terminal.

pip install bs4

**Navigation With BeautifulSoup**

Below code snippet is the HTML document which we shall use, to navigate using BeautifulSoup tags with this code snippet as reference.

ht\_doc = """

<html><head><title>Geeks For Geeks</title></head>

<body>

<p class="title"><b>most viewed courses in GFG,its all free</b></p>

<p class ="prog">Top 5 Popular Programming Languages</p>

<a href="https://www.geeksforgeeks.org/java-programming-examples/" **\**

class="prog" id="link1">Java</a>

<a href="https://www.geeksforgeeks.org/cc-programs/" class="prog" **\**

id="link2">c/c++</a>

<a href="https://www.geeksforgeeks.org/python-programming-examples/"**\**

class="prog" id="link3">Python</a>

<a href="https://https://www.geeksforgeeks.org/introduction-to-javascript/"**\**

class="prog" id="link4">Javascript</a>

<a href="https://www.geeksforgeeks.org/ruby-programming-language/" **\**

class="prog" id="link5">Ruby</a>

<p>according to an online survey. </p>

<p class="prog"> Programming Languages</p>

</body></html>

"""

Now let us navigate in all possible ways by applying BeautifulSoup in Python on the above code snippet, the most important component in Html documents are tags which may also contain other tags/strings(tag's children). BeautifulSoup provides different ways to iterate over these children, let us see all possible cases

**Navigating Downwards**

**Navigating Using Tag Names :**

**Example 1:**To get Head Tag.

Use .head to BeautifulSoup object to get the head tag in HTML document.

***Syntax :*** *(BeautifulSoup Variable).head*

**Example 2:** To get Title Tag

Use .title tag to retrieve the title of the HTML document embedded in BeautifulSoup variable

***Syntax :*** *(BeautifulSoup Variable).title*

**Code:**

soup = BeautifulSoup(ht\_doc, 'html.parser')

print(soup.head)

print(soup.title)

**Output:**

<head><title>Geeks For Geeks</title></head>

<title>Geeks For Geeks</title>

**Example 3:** To get a specific tag.

We can retrieve some specific tags like the first <b> tag in the body tag

***Syntax :*** *(BeautifulSoup Variable).body.b*

Using tag name as an attribute will get you the first name of that name

***Syntax:*** *(BeautifulSoup Variable).(tag attribute)*

By using find\_all, we can get all contents associated with the attribute

***Syntax:*** *(BeautifulSoup Variable).find\_all(tag value)*

**Code:**

soup = BeautifulSoup(ht\_doc, 'html.parser')

*# retrieving b tag element*

print(soup.body.b)

*# retrieving a tag element from BeautifulSoup assigned variable*

print(soup.a)

*# retrieving all elements tagged with a in ht\_doc*

print(soup.find\_all("a"))

**Output:**

*<b>most viewed courses in GFG,its all free</b>*

*<a class="prog" href="https://www.geeksforgeeks.org/java-programming-examples/" id="link1">Java</a>*

*[<a class="prog" href="https://www.geeksforgeeks.org/java-programming-examples/" id="link1">Java</a>,*

*<a class="prog" href="https://www.geeksforgeeks.org/cc-programs/" id="link2">c/c++</a>,*

*<a class="prog" href="https://www.geeksforgeeks.org/python-programming-examples/" id="link3">Python</a>,*

*<a class="prog" href="https://https://www.geeksforgeeks.org/introduction-to-javascript/" id="link4">Javascript</a>,*

*<a class="prog" href="https://www.geeksforgeeks.org/ruby-programming-language/" id="link5">Ruby</a>]*

**Example 4:** Contents and .children

We can get tags children in a list by using .contents.

***Syntax:*** *(BeautifulSoup Variable).contents*

**Code:**

soup = BeautifulSoup(ht\_doc, 'html.parser')

*# assigning head tag of BeautifulSoup variable*

hTag = soup.head

print(hTag)

*# retrieving contents of BeautifulSoup variable*

print(hTag.contents)

**Output:**

<head><title>Geeks For Geeks</title></head>

[<title>Geeks For Geeks</title>]

**Example 5:** .descendants

The .descendants attribute allows you to iterate over all of a tag’s children, recursively −its direct children and the children of its direct children and so on...

***Syntax:*** *(Variable assigned from BeautifulSoup Variable).descendants*

**Code:**

*# embedding html document inyto BeautifulSoup variable*

soup = BeautifulSoup(ht\_doc, 'html.parser')

*# assigning head element of BeautifulSoup-assigned Variable*

htag=soup.head

*# iterating through child in descendants of htag variable*

**for** child **in** htag.descendants:

print(child)

**Output :**

<title>Geeks For Geeks</title>

Geeks For Geeks

**Example 6:**.string

If the tag has only one child, and that child is a NavigableString, the child is made available as .string

However, if a tag contains more than one thing, then it’s not clear what .string should refer to, so .string is defined to None, we can see this practical working in below code.

soup = BeautifulSoup(ht\_doc, 'html.parser')

htag = soup.head

print(htag.string)

**Output:**

Geeks For Geeks

**Example 7: .strings and stripped\_strings**

If there’s more than one thing inside a tag, you can still look at just the strings. Use the .strings generator.

soup = BeautifulSoup(ht\_doc, 'html.parser')

**for** string **in** soup.strings :

print(repr(string))

For removal of extra whitespaces, we use .stripped\_strings generator :

*# embedding HTML document in BeautifulSoup-assigned variable*

soup = BeautifulSoup(ht\_doc, 'html.parser')

*# iterating through string in stripped\_strings of*

*# BeautifulSoup assigned variable*

**for** string **in** soup.stripped\_strings :

print(repr(string))

**Navigating Upwards Through BeautifulSoup :**

If we consider a “family tree” analogy, every tag and every string has a parent: the tag that contains it:

**Example 1:** .parent.

.parent tag is used for retrieving the element's parent element

***Syntax :*** *(BeautifulSoup Variable).parent*

**from** **bs4** **import** BeautifulSoup

soup = BeautifulSoup(ht\_doc, 'html.parser')

*# embedding html document*

Itag = soup.title

*# assigning title tag of BeautifulSoup-assigned variable*

*# to print parent element in Itag variable*

print(Itag.parent)

htmlTag = soup.html

print(type(htmlTag.parent))

print(soup.parent)

**Output:**

<head><title>Geeks For Geeks</title></head>

<class 'bs4.BeautifulSoup'>

None

**Example 2:** .parents

For iterating all over the parent elements, .parents tag can be used :

***Syntax :****(BeautifulSoup Variable).parents*

*# embedding html doc into BeautifulSoup*

soup = BeautifulSoup(ht\_doc, 'html.parser')

*# embedding a tag into link variable*

link = soup.a

print(link)

*# iterating through parent in link variable*

**for** parent **in** link.parents :

*# printing statement for Parent is empty case*

**if** parent **is** **None** :

print(parent)

**else** :

print(parent.name)

**Navigating Sideways With BeautifulSoup**

.next\_sibling and .previous\_sibling are the tags that are used for navigating between page elements that are on same level of the parse tree.

***Syntax:***

*(BeautifulSoup Variable).(tag attribute).next\_sibling*

*(BeautifulSoup Variable).(tag attribute).previous\_sibling*

**Code:**

**from** **bs4** **import** BeautifulSoup

sibling\_soup = BeautifulSoup("<a><b>Geeks For Geeks</b><c><strong>The **\**

Biggest Online Tutorials Library, It's all Free</strong></b></a>")

*# to retrieve next sibling of b tag*

print(sibling\_soup.b.next\_sibling)

*# for retrieving previous sibling of c tag*

print(sibling\_soup.c.previous\_sibling)

**Output:**

<c><strong>The Biggest Online Tutorials Library, It's all Free</strong></c>

<b>Geeks For Geeks</b>

**descendants** generator is provided by Beautiful Soup which is a web scraping framework for Python. Web scraping is the process of extracting data from the website using automated tools to make the process faster. The *.contents* and *.children* attribute only consider a tag’s direct children. The descendants generator is used to iterate over all of the tag’s children, recursively. Each child is going to be the tag element for the elements and NavigableString for the strings.**Syntax:**

tag.descendants

**Below given examples explain the concept of descendants generator in Beautiful Soup.**   
**Example 1:** In this example, we are going to get the descendants of an element.

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Create the document*

doc = "<body><b> Hello world </b><body>"

*# Initialize the object with the document*

soup = BeautifulSoup(doc, "html.parser")

*# Get the body tag*

tag = soup.body

*# Print all the descendants of tag*

**for** descendant **in** tag.descendants:

print(descendant)

**Output:**

<b> Hello world </b>

Hello world

<body></body>

**Example 2:** In this example, we are going to see the type of descendants.

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Create the document*

doc = "<body><b> Hello world </b><body>"

*# Initialize the object with the document*

soup = BeautifulSoup(doc, "html.parser")

*# Get the body tag*

tag = soup.body

*# Print the type of the descendants of tag*

**for** descendant **in** tag.descendants:

print(type(descendant))

**Output:**

<class 'bs4.element.Tag'>

<class 'bs4.element.NavigableString'>

<class 'bs4.element.Tag'>

**Python BeautifulSoup - find all class**

**Prerequisite:-** [Requests](https://www.geeksforgeeks.org/python-requests-tutorial/), [BeautifulSoup](https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/)

The task is to write a program to find all the classes for a given Website URL. In Beautiful Soup there is no in-built method to find all classes.

**Module needed:**

* [**bs4**](https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/)**:** Beautiful Soup(bs4) is a Python library for pulling data out of HTML and XML files. This module does not come built-in with Python. To install this type the below command in the terminal.

pip install bs4

* [**requests**](https://www.geeksforgeeks.org/python-requests-tutorial/)**:**  Requests allows you to send HTTP/1.1 requests extremely easily. This module also does not come built-in with Python. To install this type the below command in the terminal.

pip install requests

**Methods #1:**Finding the class in a given HTML document.

**Approach:**

* Create an HTML doc.
* Import module.
* Parse the content into BeautifulSoup.
* Iterate the data by class name.

**Code:**

*# html code*

html\_doc = """<html><head><title>Welcome to geeksforgeeks</title></head>

<body>

<p class="title"><b>Geeks</b></p>

<p class="body">geeksforgeeks a computer science portal for geeks

</body>

"""

*# import module*

**from** **bs4** **import** BeautifulSoup

*# parse html content*

soup = BeautifulSoup( html\_doc , 'html.parser')

*# Finding by class name*

soup.find( class\_ = "body" )

**Output:**

<p class="body">geeksforgeeks a computer science portal for geeks

</p>

**Methods #2:**Below is the program to find all class in a URL.

**Approach:**

* Import module
* Make requests instance and pass into URL
* Pass the requests into a Beautifulsoup() function
* Then we will iterate all tags and fetch class name

**Code:**

*# Import Module*

**from** **bs4** **import** BeautifulSoup

**import** **requests**

*# Website URL*

URL = 'https://www.geeksforgeeks.org/'

*# class list set*

class\_list = set()

*# Page content from Website URL*

page = requests.get( URL )

*# parse html content*

soup = BeautifulSoup( page.content , 'html.parser')

*# get all tags*

tags = {tag.name **for** tag **in** soup.find\_all()}

*# iterate all tags*

**for** tag **in** tags:

*# find all element of tag*

**for** i **in** soup.find\_all( tag ):

*# if tag has attribute of class*

**if** i.has\_attr( "class" ):

**if** len( i['class'] ) != 0:

class\_list.add(" ".join( i['class']))

print( class\_list )

**BeautifulSoup - Search by text inside a tag**

Beautifulsoup is a powerful python module used for web scraping. This article discusses how a specific text can be searched inside a given tag.

**INTRODUCTION:**

BeautifulSoup is a Python library for parsing HTML and XML documents. It provides a simple and intuitive API for navigating, searching, and modifying the parse tree of an HTML or XML document. It is designed to make it easy to extract data from web pages, and can be used for web scraping, data mining, and other types of data extraction tasks. It is built on the top of the powerful lxml parser, which is very fast and efficient.

BeautifulSoup is often used in combination with other Python libraries such as requests or Selenium to automate the process of downloading and parsing web pages. It can handle malformed or incomplete HTML, which is common in the real world, and provides several methods for searching for tags based on their contents, such as find(), find\_all(), and select(). It also has built-in support for searching for tags using regular expressions.

Overall, BeautifulSoup is a valuable tool for anyone working with HTML or XML data, providing a simple and intuitive API for parsing and manipulating data, and it's widely used in web scraping and data mining tasks.

**Approach**

* Import module
* Pass the URL
* Request page
* Specify the tag to be searched
* For Search by text inside tag we need to check condition to with help of string function.
* The string function will return the text inside a tag.
* When we will navigate tag then we will check the condition with the text.
* Return text

We will see search text inside a tag by two method.

**Method 1: iterative**

This method uses for loop for to search for the text.

**Example**

**from** **bs4** **import** BeautifulSoup

**import** **requests**

*# sample web page*

sample\_web\_page = 'https://www.geeksforgeeks.org/caching-page-tables/'

*# call get method to request that page*

page = requests.get(sample\_web\_page)

*# with the help of beautifulSoup and html parser create soup*

soup = BeautifulSoup(page.content, "html.parser")

child\_soup = soup.find\_all('strong')

text = 'page table base register (PTBR)'

*# we will search the tag with in which text is same as given text*

**for** i **in** child\_soup:

**if**(i.string == text):

print(i)

**Output**

*<strong>page table base register (PTBR)</strong>*

**Method 2: Using lambda**

It is a one liner alternative of the above example.

**Example**

**from** **bs4** **import** BeautifulSoup

**import** **requests**

*# sample web page*

sample\_web\_page = 'https://www.geeksforgeeks.org/caching-page-tables/'

*# call get method to request that page*

page = requests.get(sample\_web\_page)

*# with the help of beautifulSoup and html parser create soup*

soup = BeautifulSoup(page.content, "html.parser")

text = 'CS Theory Course'

*# Search by text with the help of lambda function*

gfg = soup.find\_all(**lambda** tag: tag.name == "strong" **and** text **in** tag.text)

print(gfg)

**Output**

*[<strong>CS Theory Course</strong>]*

**IMPORTANTS POINTS:**

**Here are some important points to consider when using BeautifulSoup to search for text inside a tag:**

BeautifulSoup provides several methods for searching for tags based on their contents, such as find(), find\_all(), and select().  
The find\_all() method returns a list of all tags that match a given filter, while the find() method returns the first tag that matches the filter.  
You can use the text keyword argument to search for tags that contain specific text.  
You can also use regular expressions to search for text inside a tag.

**Scrape Google Search Results using Python BeautifulSoup**

In this article, we are going to see how to Scrape Google Search Results using Python BeautifulSoup.

**Module Needed:**

* **bs4:** Beautiful Soup(bs4) is a Python library for pulling data out of HTML and XML files. This module does not come built-in with Python. To install this type the below command in the terminal.

pip install bs4

* **requests:** Requests allows you to send HTTP/1.1 requests extremely easily. This module also does not come built-in with Python. To install this type the below command in the terminal.

pip install requests

**Approach:**

* Import the beautifulsoup and request libraries.
* Make two strings with the default Google search URL, **'https://google.com/search?q='**and our customized search keyword.
* Concatenate these two strings to get our search URL.
* Fetch the URL data using **requests.get(url),**store it in a variable, **request\_result**.
* Create a string and store the result of our fetched request, using **request\_result.text.**
* Now we use BeautifulSoup to analyze the extracted page. We can simply create an object to perform those operations but beautifulsoup comes with a lot of in-built features to scrape the web. We have created a soup object first using beautifulsoup from the request-response
* We can do **soup.find.all(h3)**to grab all major headings of our search result, Iterate through the object and print it as a string.

**Example 1:** Below is the implementation of the above approach.

*# Import the beautifulsoup*

*# and request libraries of python.*

**import** **requests**

**import** **bs4**

*# Make two strings with default google search URL*

*# 'https://google.com/search?q=' and*

*# our customized search keyword.*

*# Concatenate them*

text= "geeksforgeeks"

url = 'https://google.com/search?q=' + text

*# Fetch the URL data using requests.get(url),*

*# store it in a variable, request\_result.*

request\_result=requests.get( url )

*# Creating soup from the fetched request*

soup = bs4.BeautifulSoup(request\_result.text,

"html.parser")

print(soup)

Let's We can do **soup.find.all(h3)**to grab all major headings of our search result, Iterate through the object and print it as a string.

*# soup.find.all( h3 ) to grab*

*# all major headings of our search result,*

heading\_object=soup.find\_all( 'h3' )

*# Iterate through the object*

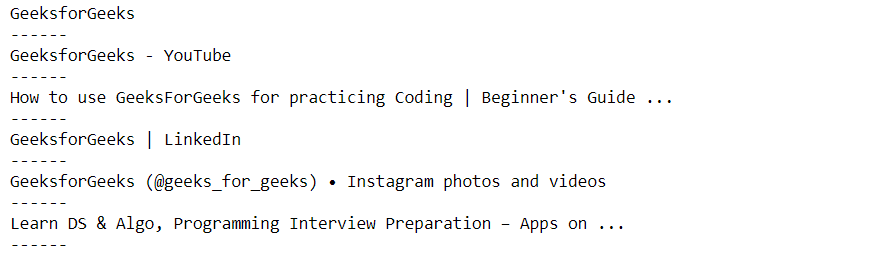
*# and print it as a string.*

**for** info **in** heading\_object:

print(info.getText())

print("------")

**Output:**



**Example 2:**Below is the implementation. In the form of extracting the city temperature using Google search:

*# import module*

**import** **requests**

**import** **bs4**

*# Taking thecity name as an input from the user*

city = "Imphal"

*# Generating the url*

url = "https://google.com/search?q=weather+in+" + city

*# Sending HTTP request*

request\_result = requests.get( url )

*# Pulling HTTP data from internet*

soup = bs4.BeautifulSoup( request\_result.text

, "html.parser" )

*# Finding temperature in Celsius.*

*# The temperature is stored inside the class "BNeawe".*

temp = soup.find( "div" , class\_='BNeawe' ).text

print( temp )

**Get tag name using Beautifulsoup in Python**

**Prerequisite:** [Beautifulsoup Installation](https://www.geeksforgeeks.org/beautifulsoup-installation-python/" \t "_blank) **Name** property is provided by Beautiful Soup which is a web scraping framework for Python. Web scraping is the process of extracting data from the website using automated tools to make the process faster. Name object corresponds to the name of an XML or HTML tag in the original document. **Syntax:**

tag.name

**Parameters:** This function doesn't accept any parameter. **Implementation:** **Example 1:** Program to extract name of a XML tag.

*# Import module*

**from** **bs4** **import** BeautifulSoup

*# Initialize the object with a XML*

soup = BeautifulSoup('''

<root>

<name\_of\_tag>the first strong tag</name\_of\_tag>

</root>

''', "lxml")

*# Get the tag*

tag = soup.name\_of\_tag

*# Get the tag name*

name = tag.name

*# Print the output*

print(name)

**Output:**

name\_of\_tag

**Example 2:** Program that explains the above functionality for a HTML tag.

*# Import module*

**from** **bs4** **import** BeautifulSoup

*# Initialize the object with a HTML page*

soup = BeautifulSoup('''

<html>

<h2> Heading 1 </h2>

<h1> Heading 2 </h1>

</html>

''', "lxml")

*# Get the whole h2 tag*

tag = soup.h2

*# Get the name of the tag*

name = tag.name

*# Print the output*

print(name)

**Output:**

h2

**Extracting an attribute value with beautifulsoup in Python**

Last Updated : 29 Dec, 2020

**Prerequisite:** [Beautifulsoup Installation](https://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-beautiful-soup)

Attributes are provided by Beautiful Soup which is a web scraping framework for Python. Web scraping is the process of extracting data from the website using automated tools to make the process faster. A tag may have any number of attributes. For example, the tag <b class="active"> has an attribute “class” whose value is “active”. We can access a tag’s attributes by treating it like a dictionary.

**Syntax:**

tag.attrs

**Implementation:**  
**Example 1:** Program to extract the attributes using attrs approach.

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Initialize the object with a HTML page*

soup = BeautifulSoup('''

<html>

<h2 class="hello"> Heading 1 </h2>

<h1> Heading 2 </h1>

</html>

''', "lxml")

*# Get the whole h2 tag*

tag = soup.h2

*# Get the attribute*

attribute = tag.attrs

*# Print the output*

print(attribute)

**Output:**

{'class': ['hello']}

**Example 2:** Program to extract the attributes using dictionary approach.

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Initialize the object with a HTML page*

soup = BeautifulSoup('''

<html>

<h2 class="hello"> Heading 1 </h2>

<h1> Heading 2 </h1>

</html>

''', "lxml")

*# Get the whole h2 tag*

tag = soup.h2

*# Get the attribute*

attribute = tag['class']

*# Print the output*

print(attribute)

**Output:**

['hello']

**Example 3:** Program to extract the multiple attribute values using dictionary approach.

*# Import Beautiful Soup*

**from** **bs4** **import** BeautifulSoup

*# Initialize the object with a HTML page*

soup = BeautifulSoup('''

<html>

<h2 class="first second third"> Heading 1 </h2>

<h1> Heading 2 </h1>

</html>

''', "lxml")

*# Get the whole h2 tag*

tag = soup.h2

*# Get the attribute*

attribute = tag['class']

*# Print the output*

print(attribute)

**Output:**

['first', 'second', 'third']

**How to insert a new tag into a BeautifulSoup object?**

Last Updated : 16 Mar, 2021

In this article, we will see how to insert a new tag into a BeautifulSoup object. See the below examples to get a better idea about the topic.

**Example:**

***HTML\_DOC :***

*"""*

*<html>*

*<head>*

*<title> Table Data </title>*

*</head>*

*<body>*

*<div> This is sample div 1 </div>*

*<div> This is sample div 2 </div>*

*</body>*

*</html>*

*"""*

***new\_tag :*** *<div> This is new div </div>*

***Modified BeautifulSoup Object :***

*"""*

*<html>*

*<head>*

*<title> Table Data </title>*

*</head>*

*<body>*

*<div> This is sample div 1 </div>*

*<div> This is sample div 2 </div>*

*<div> This is new div </div>*

*</body>*

*</html>*

*"""*

**Required Modules:**

**BeautifulSoup (bs4):** It is a Python library for pulling data out of HTML and XML files. This module does not come built-in with Python. Run the following command in the terminal to install this library-

pip install bs4

or

pip install beautifulsoup4

**Creating a new tag using new\_tag() method :**

A new tag can be created by calling BeautifulSoup's inbuilt function **new\_tag()**.

**Inserting a new tag using the append() method :**

The new tag is appended to the end of the parent tag.

*# Import Module*

**from** **bs4** **import** BeautifulSoup

*# HTML Document*

HTML\_DOC = """

<html>

<head>

<title> Add new Tag </title>

</head>

<body>

<div> This is sample div 1 </div>

<div> This is sample div 2 </div>

</body>

</html>

"""

*# Function to append new tag*

**def** addTag(html):

*# parse html content*

soup = BeautifulSoup(html, "html.parser")

*# create new tag*

*# Here we are creating a new div*

new\_div = soup.new\_tag("div")

*# Adding content to div*

new\_div.string = " This is new div "

*# Appending new div to html tree*

soup.html.body.append(new\_div)

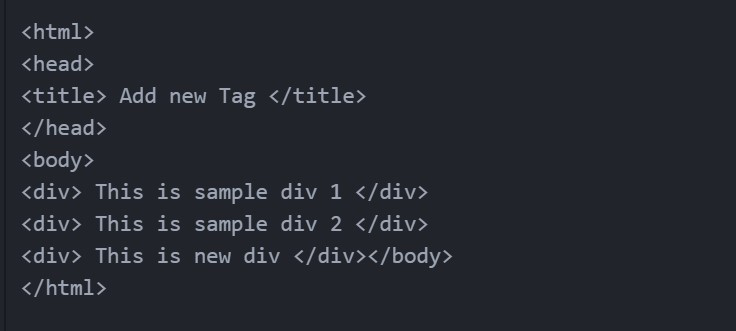
*# Printing the modified object*

print(soup)

*# Function Call*

addTag(HTML\_DOC)

**Output:**



**Inserting a new tag using insert() method :**

Using this method, the new tag is not appended to the end of the parent tag but is inserted at a given numeric position. It works the same as the[.insert()](https://www.geeksforgeeks.org/python-list-insert/) method of the Python list. For example, if we want to insert the new div between div 1 and div 2, we can use

soup.html.body.insert(2, new\_div)

This would insert the new div at position 2 i.e, between the old 2 divs.

*# Import Module*

**from** **bs4** **import** BeautifulSoup

*# HTML Document*

HTML\_DOC = """

<html>

<head>

<title> Add new Tag </title>

</head>

<body>

<div> This is sample div 1 </div>

<div> This is sample div 2 </div>

</body>

</html>

"""

*# Function to inset new tag*

**def** addTag(html):

*# parse html content*

soup = BeautifulSoup(html, "html.parser")

*# create new tag*

*# Here we are creating a new div*

new\_div = soup.new\_tag("div")

*# Adding content to div*

new\_div.string = " This is new div "

*# Inserting new div to html tree*

*# Here, 2 represents the position*

*# where we want to insert the new tag*

soup.html.body.insert(2, new\_div)

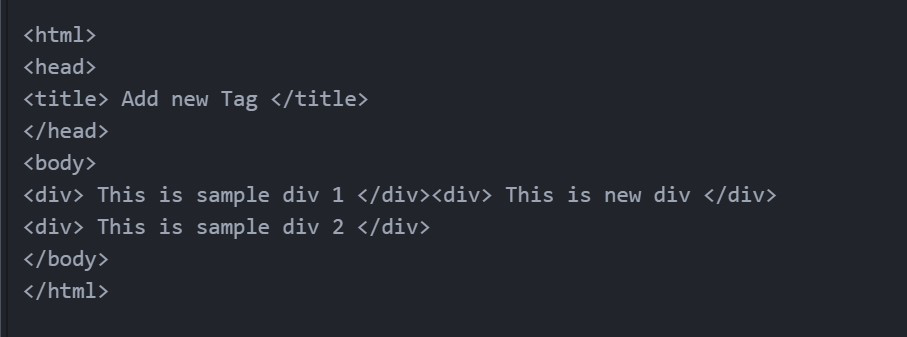
*# Printing the modified object*

print(soup)

*# Function Call*

addTag(HTML\_DOC)

**Output:**



**Inserting a new tag using insert\_before() method :**

insert\_before() method is used to insert a new tag just before the given tag.

*# Import Module*

**from** **bs4** **import** BeautifulSoup

*# HTML Document*

HTML\_DOC = """

<html>

<head>

<title> Add new Tag </title>

</head>

<body>

<div> This is sample div 1 </div>

<div> This is sample div 2 </div>

</body>

</html>

"""

*# Function to insert new tag before given tag*

**def** addTag(html):

*# parse html content*

soup = BeautifulSoup(html, "html.parser")

*# create new tag*

*# Here we are creating a new div*

new\_div\_before = soup.new\_tag("div")

*# Adding content to div*

new\_div\_before.string = " This is new div before div 1 "

*# Inserting new tag before div 1*

soup.html.body.div.insert\_before(new\_div\_before)

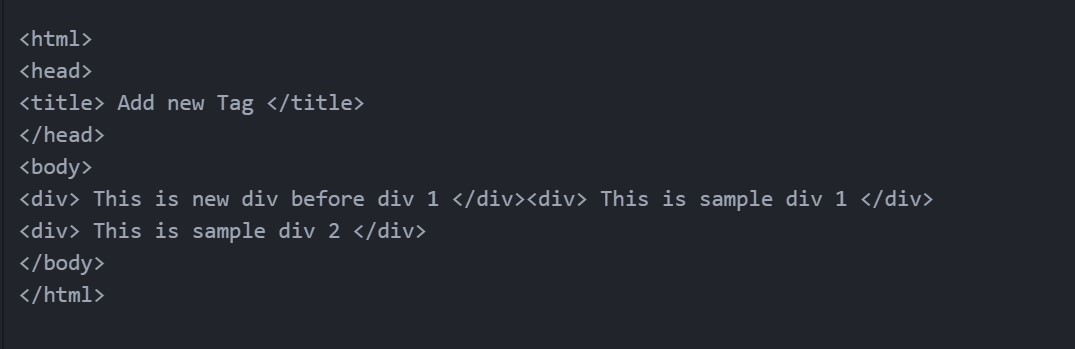
*# Printing the modified object*

print(soup)

*# Function Call*

addTag(HTML\_DOC)

**Output:**



**Inserting a new tag using insert\_after() method :**

insert\_after() method is used to insert a new tag just after the given tag.

*# Import Module*

**from** **bs4** **import** BeautifulSoup

*# HTML Document*

HTML\_DOC = """

<html>

<head>

<title> Add new Tag </title>

</head>

<body>

<div> This is sample div 1 </div>

<div> This is sample div 2 </div>

</body>

</html>

"""

*# Function to insert new tag after given tag*

**def** addTag(html):

*# parse html content*

soup = BeautifulSoup(html, "html.parser")

*# create new tag*

*# Here we are creating a new div*

new\_div\_after = soup.new\_tag("div")

*# Adding content to div*

new\_div\_after.string = " This is new div after div 1 "

*# Inserting new tag after div 1*

soup.html.body.div.insert\_after(new\_div\_after)

*# Printing the modified object*

print(soup)

*# Function Call*

addTag(HTML\_DOC)

**How to Remove tags using BeautifulSoup in Python?**

Last Updated : 07 Feb, 2023

**Prerequisite-**[Beautifulsoup module](https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/)

In this article, we are going to draft a python script that removes a tag from the tree and then completely destroys it and its contents. For this, decompose() method is used which comes built into the module.

**Syntax:**

Beautifulsoup.Tag.decompose()

Tag.decompose() removes a tag from the tree of a given HTML document, then completely destroys it and its contents.

**Implementation:**

**Example 1:**

*# import module*

**from** **bs4** **import** BeautifulSoup

*# URL for scraping data*

markup = '<a href="https://www.geeksforgeeks.org/">Welcome to <i>geeksforgeeks.com</i></a>'

*# get URL html*

soup = BeautifulSoup(markup, 'html.parser')

*# display before decompose*

print("Before Decompose")

print(soup.a)

*# decomposing the*

*# soup data*

new\_tag = soup.a.decompose()

print("After decomposing:")

print(new\_tag)

**Output:**

*Before Decompose*

*<a href="https://www.geeksforgeeks.org/">Welcome to <i>geeksforgeeks.com</i></a>*

*After decomposing:*

*None*

**Example 2:**Implementation of given URL to scrape the HTML document.

*# import module*

**from** **bs4** **import** BeautifulSoup

**import** **requests**

*# Get URL html*

*# Scraping the data from*

*# Html doc*

url = 'https://www.geeksforgeeks.org/'

reqs = requests.get(url)

soup = BeautifulSoup(reqs.text, 'html.parser')

*# Before decomposing*

print("Before Decomposing")

print(soup)

*# decompose the soup*

result = soup.decompose()

print("After decomposing:")

print(result)

**Remove all style, scripts, and HTML tags using BeautifulSoup**

Last Updated : 03 Jun, 2024

**Prerequisite:** [BeautifulSoup](https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup), [Requests](https://www.geeksforgeeks.org/python-requests-tutorial)

Beautiful Soup is a Python library for pulling data out of HTML and XML files. In this article, we are going to discuss how to remove all style, scripts, and HTML tags using beautiful soup.

**Required Modules:**

* **bs4:** Beautiful Soup (bs4) is a python library primarily used to extract data from HTML, XML, and other markup languages. It's one of the most used libraries for Web Scraping.   
  Run the following command in the terminal to install this library-

pip install bs4

* **requests:**This library is used for making HTTP requests in python.  
  Run the following command in the terminal to install this library-

pip install requests

**Approach:**

* Import bs4 library
* Create an HTML doc
* Parse the content into a [BeautifulSoup object](https://www.geeksforgeeks.org/beautifulsoup-object-python-beautifulsoup)
* Iterate over the data to remove the tags from the document using [decompose()](https://www.geeksforgeeks.org/how-to-remove-tags-using-beautifulsoup-in-python) method
* Use [stripped\_strings()](https://www.crummy.com/software/BeautifulSoup/bs4/doc/" \l "strings-and-stripped-strings" \t "_blank) method to retrieve the tag content
* Print the extracted data

**Implementation:**

*# Import Module*

**from** **bs4** **import** BeautifulSoup

*# HTML Document*

HTML\_DOC = """

<html>

<head>

<title> Geeksforgeeks </title>

<style>.call {background-color:black;} </style>

<script>getit</script>

</head>

<body>

is a

<div>Computer Science portal.</div>

</body>

</html>

"""

*# Function to remove tags*

**def** remove\_tags(html):

*# parse html content*

soup = BeautifulSoup(html, "html.parser")

**for** data **in** soup(['style', 'script']):

*# Remove tags*

data.decompose()

*# return data by retrieving the tag content*

**return** ' '.join(soup.stripped\_strings)

*# Print the extracted data*

print(remove\_tags(HTML\_DOC))

**Output:**

Geeksforgeeks is a Computer Science portal.

**Removing all style, scripts, and HTML tags from an URL**

**Approach:**

* Import bs4 and requests library
* Get content from the given URL using requests instance
* Parse the content into a BeautifulSoup object
* Iterate over the data to remove the tags from the document using decompose() method
* Use stripped\_strings() method to retrieve the tag content
* Print the extracted data

**Implementation:**

*# Import Module*

**from** **bs4** **import** BeautifulSoup

**import** **requests**

*# Website URL*

URL = 'https://www.geeksforgeeks.org/data-structures/'

*# Page content from Website URL*

page = requests.get(URL)

*# Function to remove tags*

**def** remove\_tags(html):

*# parse html content*

soup = BeautifulSoup(html, "html.parser")

**for** data **in** soup(['style', 'script']):

*# Remove tags*

data.decompose()

*# return data by retrieving the tag content*

**return** ' '.join(soup.stripped\_strings)

*# Print the extracted data*

print(remove\_tags(page.content))

**BeautifulSoup - Remove the contents of tag**

Last Updated : 25 Feb, 2021

In this article, we are going to see how to remove the content tag from HTML using BeautifulSoup. BeautifulSoup is a python library used for extracting html and xml files.

**Modules needed:**

[**BeautifulSoup:**](https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/)Our primary module contains a method to access a webpage over HTTP.

**For installation run this command into your terminal:**

pip install bs4

**Approach:**

* First, we will import the required libraries.
* We will read the html file or text.
* We will feed the extracted text to the soup object.
* We will then find the required tag and then clear its element.

**Step-by-step implementation:**

**Step 1:**We will initialize the program, import the libraries and read or create the HTML doc that we want soup.

*# Importing libraries*

**from** **bs4** **import** BeautifulSoup

*# Reading the html text we want to parse*

text = "<html> <head><title> Welcome </title></head><body><h1>This is a test page</h1></body></html>"

**Step 2:**We will pass the retrieved text to the soup object and set the parser in this case we are using html parser. Other markups that can be used are xml or html5. Then we will mention the tag from which we have to remove the content.

*# creating a soup*

soup = BeautifulSoup(text,"html.parser")

*# printing the content in h1 tag*

print(f"Content of h1 tag is: **{**soup.h1**}**")

**Output:**



**Step 3:**We will use .clear function. It clears the content of the mentioned tag.

*# clearing the content of the tag*

soup.h1.clear()

*# printing the content in h1 tag after clearing*

print(f"Content of h1 tag after clearing: **{**soup.h1**}**")



**Below is the full implementation:**

*# Importing libraries*

**from** **bs4** **import** BeautifulSoup

*# Reading the html text we want to parse*

text = "<html> <head><title> Welcome </title></head><body><h1>This is a test page</h1></body></html>"

*# creating a soup*

soup = BeautifulSoup(text,"html.parser")

*# printing the content in h1 tag*

print(f"Content of h1 tag is: **{**soup.h1**}**")

*# clearing the content of the tag*

soup.h1.clear()

*# printing the content in h1 tag after clearing*

print(f"Content of h1 tag after clearing: **{**soup.h1**}**")