Experiment 2

**Web Scrapping (Wikipedia)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name:** | **:** | **Abhishek N N** | **Register No** | **:** | **20BCE1025** |
| Faculty | : | Dr. Alok Chauhan | Slot | : | L51-L52 AB1-605B |
| Course | : | Web Mining Lab | Code | : | CSE3024 |
| Programme | : | B.Tech CSE Core | Semester | : | Win – 22 - 23 |



https://en.wikipedia.org/wiki/Wikipedia

this link is used for lab experiment

**Extraction of html code**

This codes are taken from python notebooks so output will be shown next to it

import requests

r = requests.get('https://en.wikipedia.org/wiki/Wikipedia')

r.status\_code



r.headers['content-type']

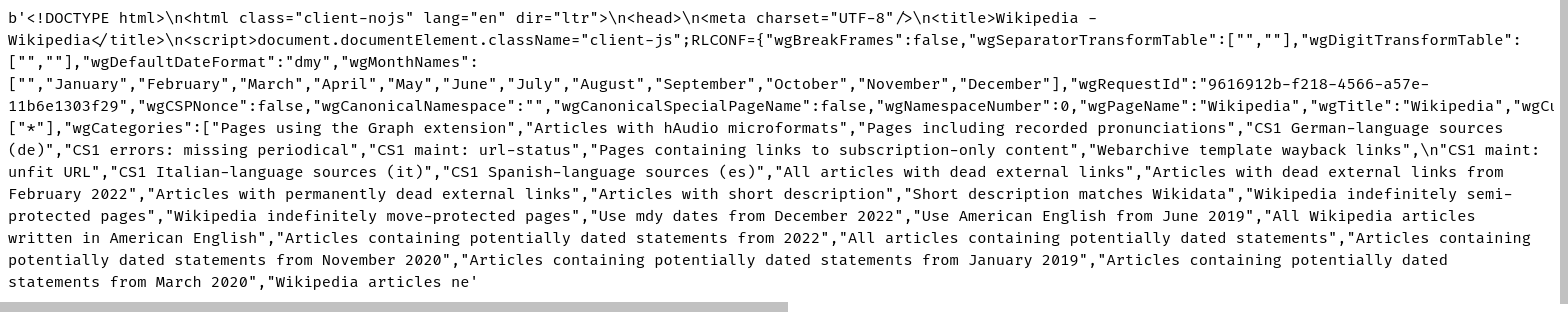


r.encoding



# output is large so printing only first 2000 characters

r.content[:2000]



**Extraction of different components of the page**

from bs4 import BeautifulSoup

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

# helper function which returns all the html tags in a list

%run helpers\_20BCE1025.ipynb

htmlTagsList = getHTMLTags()

for tags in htmlTagsList:

print("----------------------------------------------------")

print(tags[0]) # printing tag category

print("----------------------------------------------------")

for tag in tags[1:]:

fl=soup.find\_all(tag)

if len(fl)==0:

continue

print("------------")

print(tag) # printing tag name

print("------------")

for f in fl:

print(f.text.strip()) # printing tag content

print("------------")

print("----------------------------------------------------")



**Error Handling**

**Error During Fetching of Website**

When we are fetching any website content we need to aware of some of the errors that occur during fetching. These errors may be **HTTPError,** **URLError, AttributeError, or XMLParserError**. Now we will discuss each error one by one.

**HTTPError:**

HTTPError occurs when we’re performing web scraping operations on a website that is not present or not available on the server. When we provide the wrong link during requesting to the server then and we execute the program is always shows an Error **“Page Not Found”** on the terminal.

**Code:**

import requests

from urllib.error import HTTPError

def webScraping(url):

try:

response = requests.get(url)

response.raise\_for\_status()

except requests.exceptions.HTTPError as err:

print("failed")

print(err)

else:

print("success")

url1 = 'https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/'

url2 = 'https://www.geeksforgeeks.org/page-that-do-not-exist'

webScraping(url1)

print("---------------------------------------------------------")

webScraping(url2)

**Output:**



**URLError:**

When we request the wrong website from the server it means that URL which we are given for requesting is wrong then URLError will occur. URLError always responds as a server not found an error.

**Code:**

import requests

from urllib.error import URLError

link = 'https://www.amaz.in/s/ref=nb\_sb\_ss\_ts-doa-p\_3\_3?url=search-alias%3Daps&field-keywords=basketball&sprefix=bas%2Caps%2C458&crid=3STPJQX67B7GD'

try:

response = requests.get(link)

except URLError as url\_error:

print("Server Not Found")

else:

print("There is no Error")

**Output:**

****

**AttributeError:**

The AttributeError in BeautifulSoup is raised when an invalid attribute reference is made, or when an attribute assignment fails. When during the execution of code we pass the wrong attribute to a function that attribute doesn’t have a relation with that function then AttributeError occurs.  When we try to access the Tag using BeautifulSoup from a website and that tag is not present on that website then BeautifulSoup always gives an AttributeError.

We take a good example to explain the concept of AttributeError with web scraping using BeautifulSoup:

**Code:**

import requests

import bs4

url = 'https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/'

# getting response from server

response = requests.get(url)

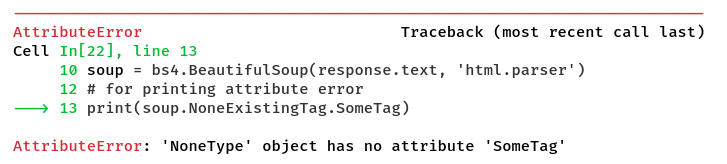
# extracting html

soup = bs4.BeautifulSoup(response.text, 'html.parser')

# for printing attribute error

print(soup.NoneExistingTag.SomeTag)

**Output:**

****

**XML Parser Error :**

We all are gone through XML parser error during coding the web scraping scripts, by the help of BeautifulSoup we parse the document into HTML very easily. If we stuck on the parser error then we easily overcome this error by using BeautifulSoup, and it is very easy to use.

When we’re parsing the HTML content from the website we generally use **‘ xml ‘**or **‘ xml-xml ‘**in the parameter of BeautifulSoup constructor. It was written as the second parameter after the HTML document.

**Code:**

import requests

import bs4

url = 'https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/'

response = requests.get(url)

soup = bs4.BeautifulSoup(response.text,'xml')

print(soup.find('div',class\_='that not present in html content'))

**Output:**

****

**HTML Parsing**

Let’s say you want to use Beautiful Soup look at a document’s <a> tags. It’s a waste of time and memory to parse the entire document and then go over it again looking for <a> tags. It would be much faster to ignore everything that wasn’t an <a> tag in the first place. The SoupStrainer class allows you to choose which parts of an incoming document are parsed. You just create a SoupStrainer and pass it in to the BeautifulSoup constructor as the parse\_only argument.

(Note that *this feature won’t work if you’re using the html5lib parser*. If you use html5lib, the whole document will be parsed, no matter what. This is because html5lib constantly rearranges the parse tree as it works, and if some part of the document didn’t actually make it into the parse tree, it’ll crash. To avoid confusion, in the examples below I’ll be forcing Beautiful Soup to use Python’s built-in parser.)

from bs4 import SoupStrainer

only\_a\_tags = SoupStrainer("a")

only\_tags\_with\_id\_link2 = SoupStrainer(id="link2")

def is\_short\_string(string):

return string is not None and len(string) < 10

only\_short\_strings = SoupStrainer(string=is\_short\_string)

html\_doc = """<html><head><title>The Dormouse's story</title></head>

<body>

<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were

<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,

<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and

<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;

and they lived at the bottom of a well.</p>

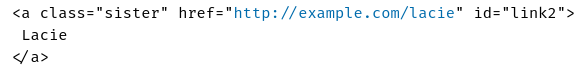
<p class="story">...</p>

"""

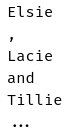
print(BeautifulSoup(html\_doc, "html.parser", parse\_only=only\_a\_tags).prettify())

****

print(BeautifulSoup(html\_doc, "html.parser", parse\_only=only\_tags\_with\_id\_link2).prettify())

****

print(BeautifulSoup(html\_doc, "html.parser", parse\_only=only\_short\_strings).prettify())

****