Web Scraping in Python
(Beautiful Soup)

DR. JASMEET SINGH,
ASSISTANT PROFESSOR,
CSED, TIET

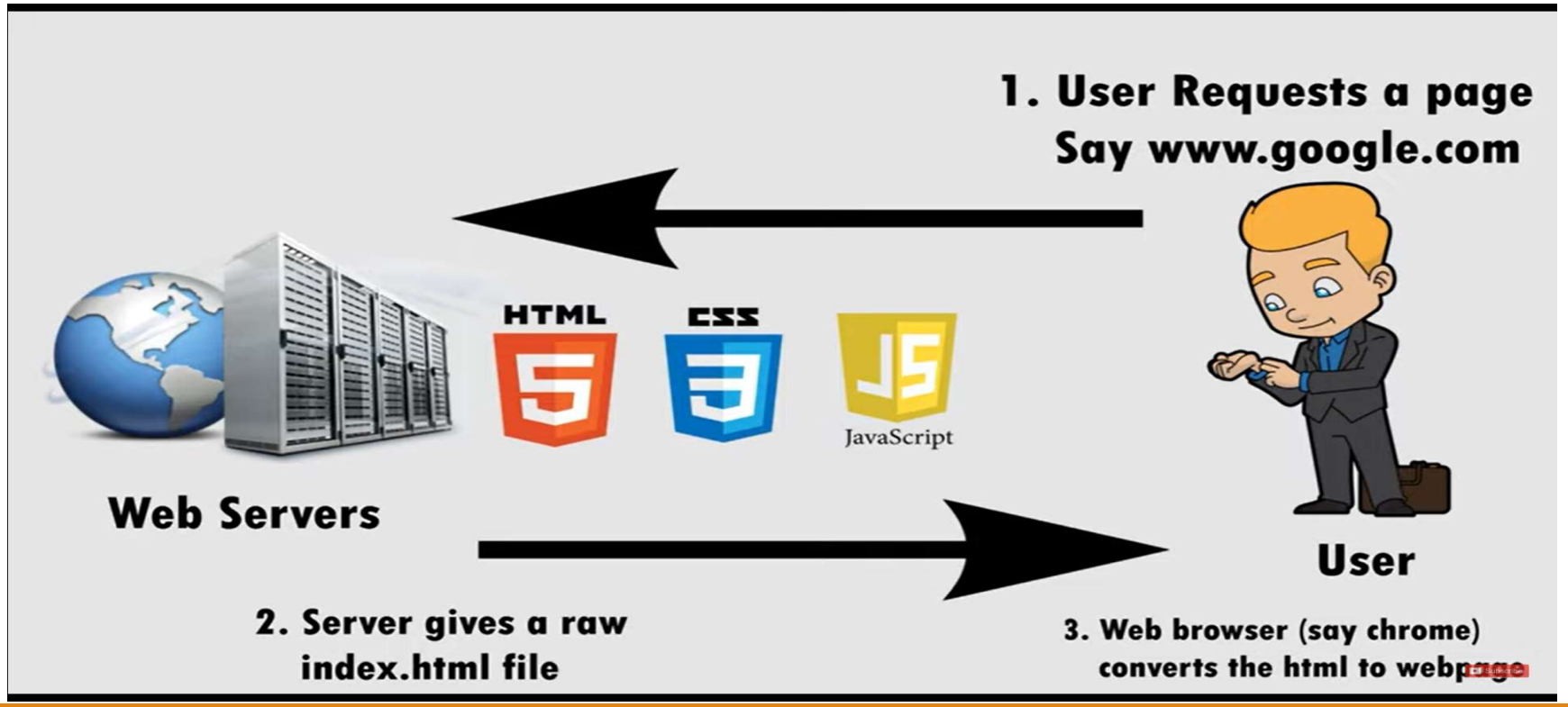
Web Scraping

- Scraping is simply a process of extracting (from various means), copying and screening of data.
- When we do scraping or extracting data or feeds from the web (like from web-pages or websites), it is termed as web-scraping.
- So, web scraping which is also known as web data extraction or web harvesting is the extraction of data from web. In short, web scraping provides a way to the developers to collect and analyze data from the internet.

Beautiful Soup

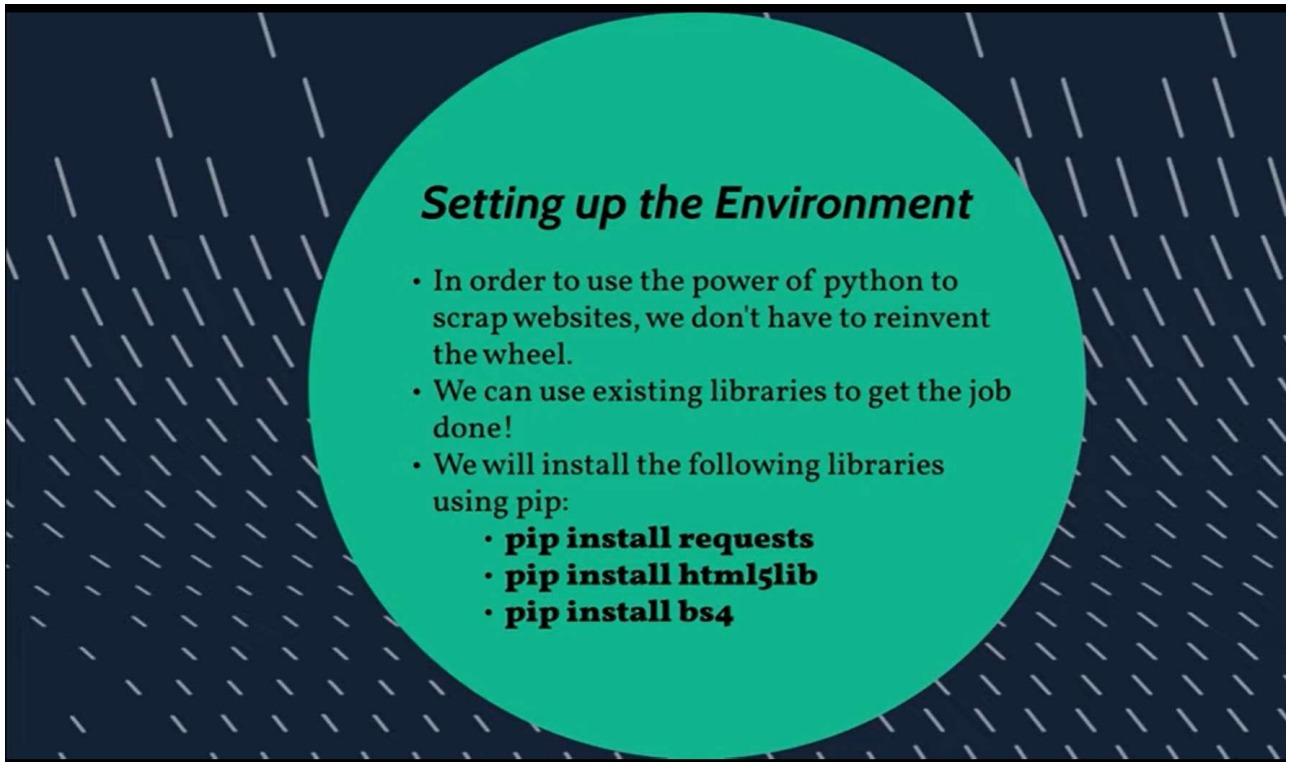
- The Beautiful Soup is a python library which is named after a Lewis Carroll poem of the same name in “Alice’s Adventures in the Wonderland”.
- Beautiful Soup is a python package and as the name suggests, parses the unwanted data and helps to organize and format the messy web data by fixing bad HTML and present to us in an easily-traversible XML structures.
- In short, Beautiful Soup is a python package which allows us to pull data out of HTML and XML documents.

Web Access Process



Web Scrap Process

Setting Up Environment

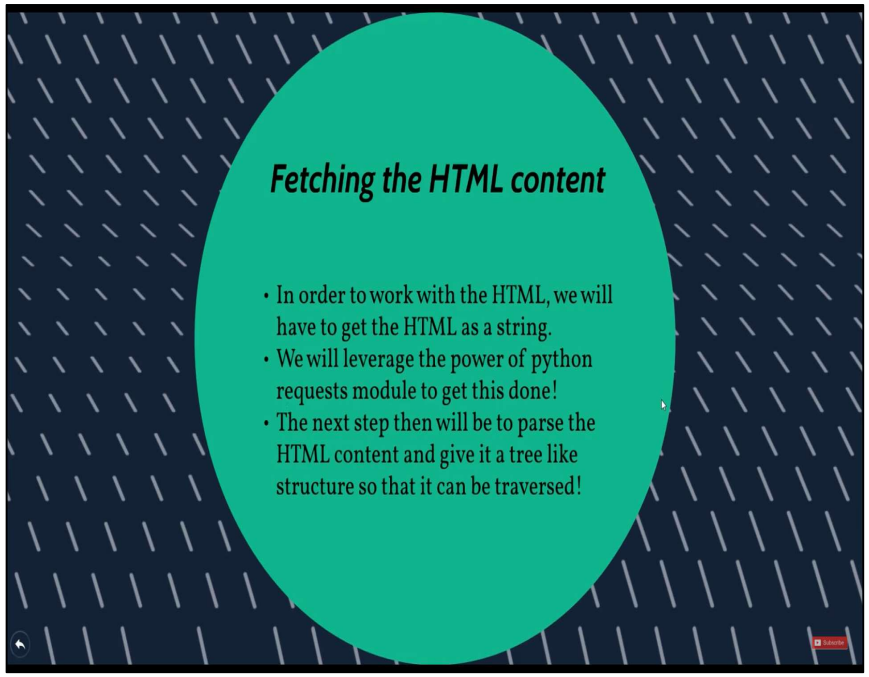


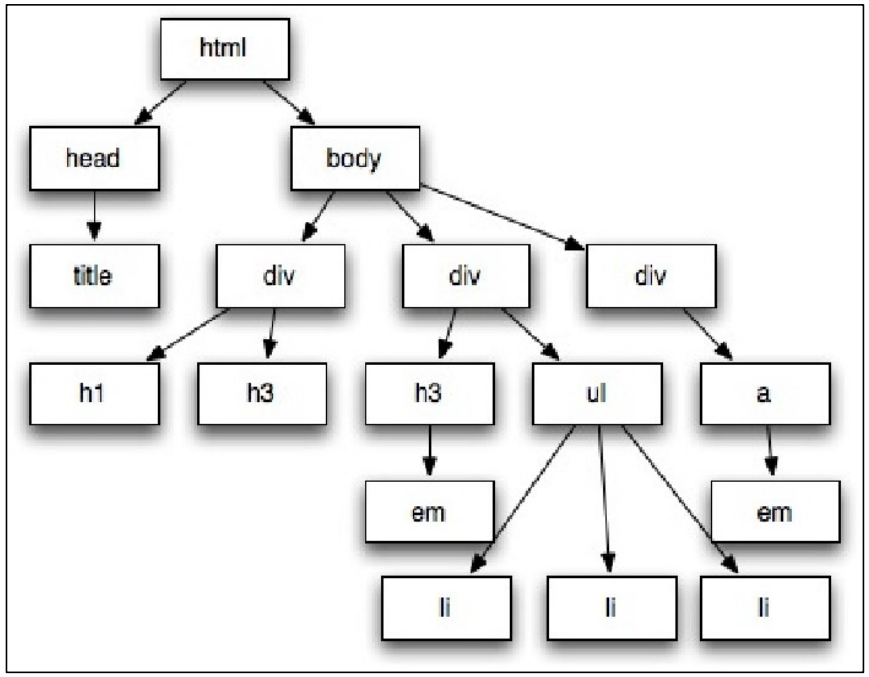
requests- helps to get html content from websites

htmllib5- helps to parse the content

bs4- helps to search and get the desired content

Fetching the HTML Content





Fetching the HTML Content

Code:

```
url = "https://www.tutorialspoint.com/index.htm"
req = requests.get(url)
html_content=req.content #.content is the content of the response in bytes
```

We can also use .text to get the content of the response in Unicode

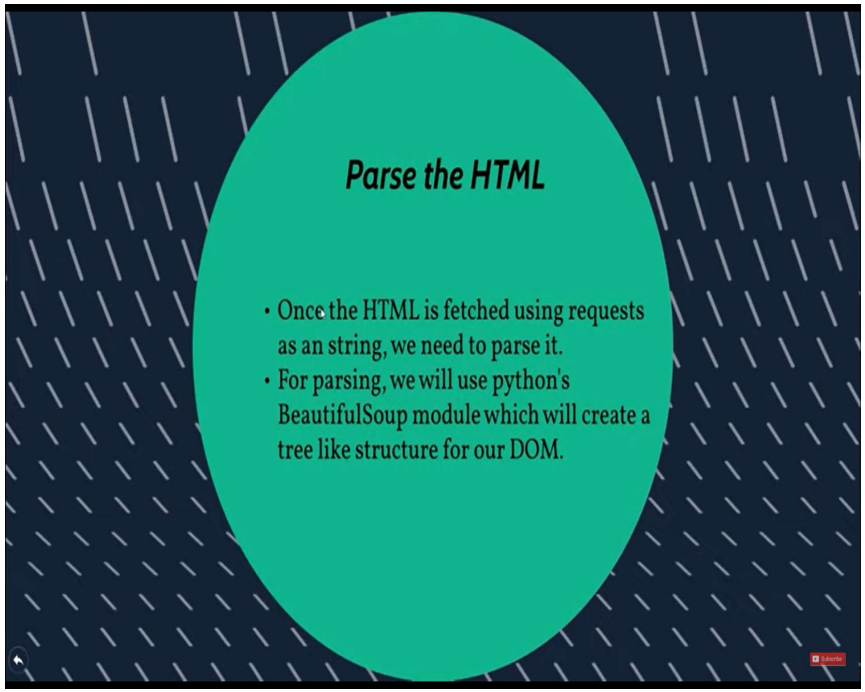
Step 2: Parsing the Content

- Parsing the Content is also called as Souping the Page
- We make use of both BeautifulSoup and Parser to parse the content of the page.
- **Code:**

```
soup=BeautifulSoup(html_content,'html.parser’)

print(soup.prettify)

#prettify will prettify the contents and print
```



Type of Objects

- When we passed a html document or string to a beautifulsoup constructor, beautifulsoup basically converts a complex html page into differe qnt python objects. Below we are going to discuss four major kinds of objects:

- Tag
- NavigableString
- BeautifulSoup

Tag Object

- A HTML tag is used to define various types of content. A tag object in BeautifulSoup corresponds to an HTML or XML tag in the actual page or document.
- For example:

soup.title has a type 'bs4.element.Tag'

- Tags contain lot of attributes and methods and two important features of a tag are its name and attributes.
- Every tag contains a name and can be accessed through '.name' as suffix. tag.name will return the type of tag it is.
- for example: soup.title.name
- However, if we change the tag name, same will be reflected in the HTML markup generated by the BeautifulSoup.

- example:soup.title.name='Strong'; soup.Strong.name
- A tag object can have any number of attributes. The tag <b class="boldest"> has an attribute 'class' whose value is "boldest".
- We can access the attributes either through accessing the keys (like accessing "class" in above example) or directly accessing through ".attrs".
- We can do all kind of modifications to our tag's attributes (add/remove/modify).
- For Example: tag=soup.div; tag.attrs

tag['Style']=2007

NavigableString Object

- The navigablestring object is used to represent the contents of a tag.
- To access the contents, use ".string" with tag.
- Example: soup.title.string
- We can replace the string with another string but you can't edit the existing string.
- Example: soup.title.string.replace_with('Hi how are you'); soup.tilte.string

BeautifulSoup Object

- BeautifulSoup is the object created when we try to scrape a web resource.
- So, it is the complete document which we are trying to scrape. Most of the time, it is treated tag object.
- soup=BeautifulSoup(html_content,'html.parser')

```
print(type(soup))

soup.name
```

Step 3: HTML Tree Traversal

Following methods are used for tree traversal

Method	Description	Example
Find	Finds the first instance of the tag (or string).	soup.find('a')
Find_all	Finds all instances of the tag(or string).	soup.find_all('a')
Find_parent	Finds the parent of the tag (or string)	soup.find_parent('a',attrs={'class':' '})
Find_parents	Finds all the parents of the tag (or string)	soup.find_parents('a',attrs={'class':' '})
find_next_sibling()	iterate over the rest of an element's siblings in the tree	soup.find_next_sibling('a',attrs={'class':' '})
find_next_siblings()	returns all the siblings that match	soup.find_next_siblings('a',attrs={'class':' '})

Simillarly we have methods find_next, find_all_next, find_previous, find_all_previous, find_next_siblings, find_next_sibling, find _previous_sibling,find_previous,siblings methods